

"Made available under NASA sponsorship
in the interest of early and wide dis-
semination of Earth Resources Survey
Program information and without liability
for any use made thereof."

T78-10856 #
* 80-10197
JSC-13894 NASA CR
160631

AS-BUILT DESIGN
FOR
ENHANCEMENT OF THE AUTOMATIC
STATUS AND TRACKING SYSTEM SOFTWARE

Job Order 71-695

(E80-10197) AS-BUILT DESIGN FOR ENHANCEMENT
OF THE AUTOMATIC STATUS AND TRACKING SYSTEM
SOFTWARE (Lockheed Electronics Co.) 146 p
HC A07/MF A01 CSCI 02C

N80-28794

Unclas
G3/43 00197

Prepared By
Lockheed Electronics Company, Inc.
System and Services Division
Houston, Texas
Contract NAS 9-15200
For
EARTH OBSERVATIONS DIVISION
SCIENCE AND APPLICATIONS DIRECTORATE



National Aeronautics and Space Administration
LYNDON B. JOHNSON SPACE CENTER
Houston, Texas

February 1978

LEC-11882

JSC-13894

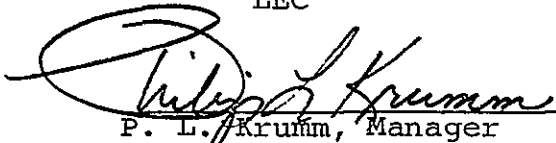
AS-BUILT DESIGN
FOR
ENHANCEMENT OF THE AUTOMATIC
STATUS AND TRACKING SYSTEM SOFTWARE

Job Order 71-695

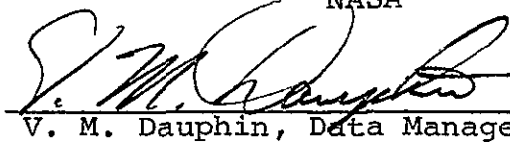
Prepared By
D. K. McCarley
L. D. Dornell

APPROVED BY

LEC


P. L. Krumm, Manager
Data Systems Department

NASA


V. M. Dauphin, Data Manager
System and Facilities Branch

Prepared By
Lockheed Electronics Company, Inc.
For
Earth Observations Division
Science and Applications Directorate
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
LYNDON B. JOHNSON SPACE CENTER
HOUSTON, TEXAS

February 1978

LEC-11882

CONTENTS

Section	Page
1. INTRODUCTION.	1-1
1.1 <u>PURPOSE AND SCOPE</u>	1-1
1.2 <u>BACKGROUND</u>	1-1
2. APPLICABLE DOCUMENTS	2-1
3. SOFTWARE MODIFICATIONS	3-1
3.1 <u>CONTROL FILES</u>	3-1
3.1.1 ASATS.BIS, BATCH RUN.	3-1
3.1.2 TY1ORDER.COM, PACKET ORDER LIST	3-6
3.1.3 RIMS CMD, RIMS TASK BUILDER COMMANDS	3-12
3.1.4 RIMS.ODL, RIMS OVERLAY STRUCTURE.	3-13
3.2 <u>NEW AND MODIFIED RIMS PROGRAMS</u>	3-15
3.2.1 PROGRAM DESCRIPTIONS.	3-15
3.2.2 PROGRAM CROSS-REFERENCE.	3-115
3.2.3 NEW BUFFER FORMATS, COMMON BLOCKS, AND TABLES	3-130

1. INTRODUCTION

1.1 PURPOSE AND SCOPE

This document makes current the existing design documentation on the LACIE Automatic Status and Tracking System (ASATS) as implemented on the PDP 11/45 computer. To accomplish this objective, it contains descriptions of all the modifications necessary to implement TIRF 77-0035, Enhancement of the ASATS Software. The complete ASATS was documented by previous publications (as listed in Section 2), and this document should be used in conjunction with those publications, superseding them where they conflict with statements made herein.

1.2 BACKGROUND

ASATS was implemented using the Regional Information Management System (RIMS), a generalized data base management system.

TIRF 77-0035 required several enhancements to be made to RIMS. They were as follows:

- a. Additional Data Base Protection - In order to prevent inadvertent destruction of portions of the data base, additional user interaction to verify the user's desire to execute the command is requested for the Delete Set (DS), Delete Record (DR), Delete Key Name (DK), and No Key (NK) commands.
- b. Null Set Detection and Control - In order to prevent production of headers for reports containing no data, the Jump Test (JT) and Label (LA) commands are implemented.
- c. Arithmetic Operators - The ability to allow arithmetic operations on fields of data is implemented for the Select Non-Key (SN), Joint Select Non-Key (JN), Display Formatted (DF), Joint Display Formatted (JF), Report (RP), Joint Report (JP), and Change Field (CF) commands.

- d. Inter Data Base Comparisons - The ability to specify arithmetic relationships between fields at different levels of the data base is implemented in the Joint Select Non-Key (JN), Joint Display Formatted (JF), and Joint Report (JP) commands. The ability to sort with fields at different levels of the data base is implemented in the Joint Sort (JS) command.
- e. Subgrouping by Field with Maximum, Minimum, Sum, and Count Functions - The ability to specify fields for which records are to be grouped by value and print field values, maximum or minimum field values, sums of field values, or counts of records for the resulting groups is implemented in the Report (RP) and Joint Report (JP) commands.

The remainder of this document identifies the changes to RIMS software required to implement the enhancements.

2. APPLICABLE DOCUMENTS

The following documents are applicable:

- a. RIMS Design Document, February 1976 (LEC-9564)
- b. RIMS Maintenance Document, October 1976 (LEC-9566)
- c. ASATS Functional Design Document, November 1976 (LEC-9861, JSC-11835)
- d. Implementation Specification for Large Area Crop Inventory Experiment (LACIE) Phase III Automatic Status and Tracking System, March 1977 (LEC-8675, JSC-11401, Rev. A)
- e. Operator's Guide for ASATS, March 1977 (LEC-10401, JSC-12729)
- f. RIMS Users Guide, April 1977 (LEC-9301, Rev. A)
- g. Detailed Design Specification for the Automatic Status and Tracking System Modifications for LACIE Procedure 1, May 1977 (LEC-10529, JSC-12885)
- h. TIRF 77-0035, May 1977
- i. ASATS Users Guide, June 1977 (LEC-10148, JSC-12535, Rev. A)
- j. As-Built Design for LACIE Phase III Automatic Status and Tracking System, June 1977 (LEC-10419, JSC-12743, Rev. A)
- k. Project Development Plan for the Enhancement of the Software of the LACIE Automatic Status and Tracking System, August 1977 (LEC-10977, Rev. A)
- l. Functional Design Specification for Enhancement of the Automatic Status and Tracking System Software, September 1977 (LEC-11199, JSC-13110)
- m. Detail Design Specification for Enhancement of the Automatic Status and Tracking System Software, November 1977 (LEC-11512, JSC-13789)

3. SOFTWARE MODIFICATIONS

3.1 CONTROL FILES

Various non-data files were changed to take advantage of the new features of RIMS or to make necessary adjustments. These files are listed in the following sections as they currently exist.

3.1.1 ASATS.BIS, BATCH RUN

```
$JOB/NAME=ASATS17/ACCOUNT=210 4/LIMIT=999/MCR
$MESSAGE ASATS BATCH STREAM VERSION 17 (FEB. 1978).
$MESSAGE FIXED TO SORT PUNCHED CARDS INTO ORDER BY TYPE.
$MESSAGE RE SURE YOU HAVE;
$MESSAGE ----- READ IN CARD DECK.
$MESSAGE ( WITH: PIP @ (210,4)UP )
$MESSAGE IN CASE OF TROUBLE, CALL:
$MESSAGE JOE EVERETTE 333-6208 (DAYS)
$MESSAGE OR JOHN CAON 483-5851 (DAYS) OR 481-0339 (NIGHTS)
$MESSAGE IF CARDS NOT O.K., ABORT THIS RUN AND RESTART
$MESSAGE WITH ANOTHER BAT ASATS AFTER READING CARDS.
$MCR PIP (210,4)DUM.TES=(210,4)ENDFILE
$MCR PIP LP=(210,4)*.TES*/I
$MESSAGE/WAIT NOW, TYPE IN CON(CR) TO CONTINUE, OR ABO(CR) TO ABORT.
$MCR PIP LP=(210,4)*.TES*/I
$MESSAGE START PREPROCESSOR
$ I CLEAN UP FILES
$MCR PIP
(210,4)DUMY.TES=(210,4)ENDFILE
(210,4)*.TES*/DE
$ I TRANSFER PROGRAMS TO (210,4) TO MAKE $RUN COMMAND WORK AS DESIRED.
$MCR PIP
(210,4)STEP1.TTK=(206,5)STEP1.TSK
(210,4)STEP3.TTK=(206,5)STEP3.TSK
(210,4)STEP5.TTK=(206,5)STEP5.TSK
(210,4)ASATS.TTK=(206,5)ASATS.TSK
(210,4)POSTP.TTK=(206,5)POSTP.TSK
(210,4)RIMS.TTK=(206,5)RIMS.TSK
$ I STEP 1. READS PPF11.CRE AND WRITES OUT THE FOLLOWING:
$ I PPF143 AND PPF144(PHASE 3 AND TY) FOR LATER PROCESSING)
$ I OTHER FILES TO LINE PRINTER
$ I SORT OPERATION WILL ALWAYS HAVE INPUT AND ALWAYS PUT OUT
$ I SOMETHING IF IT RUNS SUCCESSFULLY.
$MCR REM RSXBAT
$RUN STEP1.TTK
```

```

SMESAGE START SORT
SMCR SRT (210,4)PPF243.TES=(210,4)PPF143.TES/SIZE:80;DLSPEC.SOR
S I CAN NOW SAVE SPACE BY REMOVING INPUT TO THAT SORT.
SMCR PIP (210,4)PPF143.TES:1/DE
SMCR SRT (210,4)PPF244.TES=(210,4)PPF144.TES/SIZE:80;DLSPEC.SOR
SMCR PIP (210,4)PPF144.TES:1/DE
S I SET UP NOW FOR STEP3.
S I STEP 3 WRITES TWO FILES FOR EACH LACTE PHASE:
S I PPF35(3 AND 4) OF NON=*. 2, 3 CARDS
S I PPF33(3 AND 4) OF *. 2, 3 CARDS
S I ALL FOUR MUST EXIST IF NO MESSAGES ARE TO BE GIVEN.
SMCR PIP
(210,4)PPF333.TES=(210,4)ENDFILE
(210,4)PPF334.TES=(210,4)ENDFILE
(210,4)PPF353.TES=(210,4)ENDFILE
(210,4)PPF354.TES=(210,4)ENDFILE
SMCR REM RSXBAT
SRUN STEP3.TTK
SMCR PIP (210,4)PPF243.TES:*/DE
SMCR PIP (210,4)PPF244.TES:*/DE
SMESAGE STEP 3 OF PREPROCESSOR FINISHED
S I BEFORE DOING THE SORT, PUT UP A DUMMY OUTPUT FILE FOR IT.
SMCR PIP (210,4)PPF42P.TES=(210,4)ENDFILE
SMCR SRT (210,4)PPF42P.TES=(210,4)PPF333.TES/SIZE:80/KEYS:CN4,4:CN1,80
S I NOW CLEAN UP INPUT FILES TO THAT SORT, AND
S I UP DUMMY OUTPUTS FOR STEP 5.
SMCR PIP
(210,4)PPF333.TES:*/DE
(210,4)PPF53P.TES=(210,4)ENDFILE
(210,4)PPF55P.TES=(210,4)ENDFILE
(210,4)PPF57P.TES=(210,4)ENDFILE
SMCR PIP (210,4)PPF42P.TES:*/DE
SMCR PIP (210,4)PPF653.TES=(210,4)ENDFILE
SMCR PIP
LP:=(210,4)PPF57P.TES
(210,4)PPF57P.TES:*/DE
(210,4)PPF55P.TES:*/DE
(210,4)PHASE3.TES=(210,4)PPF53P.TES/RE
(210,4)PHASE3.TES=(210,4)PPF653.TES/AP
(210,4)PHASE3.TES=(210,4)PPF353.TES/AP
(210,4)PPF53P.TES=(210,4)ENDFILE
(210,4)PPF53P.TES:*/DE
(210,4)PPF653.TES:*/DE
(210,4)PPF353.TES:*/DE
(210,4)PPF42P.TES=(210,4)ENDFILE
SMCR SRT (210,4)PPF42P.TES=(210,4)PPF334.TES/SIZE:80/KEYS:CN4,4:CN1,80
S I NOW CLEAN UP INPUT TO THAT SORT AND GIVE DUMMY OUTPUT FOR STEP 5.
SMCR PIP
(210,4)PPF334.TES:*/DE
(210,4)PPF53P.TES=(210,4)ENDFILE
(210,4)PPF55P.TES=(210,4)ENDFILE
(210,4)PPF57P.TES=(210,4)ENDFILE
SMCR REM RSXBAT
SRUN STEP5.TTK
SMCR PIP (210,4)PPF42P.TES:*/DE
SMESAGE STEP 5 FINISHED FOR TY1.

```

ORIGINAL PAGE IS
OF POOR QUALITY


```

$ 1 SORT *.2,3 NON-SPTS BACK INTO CARD-TYPE ORDER.
$ 1 FIRST, CREATE A DUMMY SORTED FILE.
SMCR PIP [210,4]PPF654.TES=[210,4]ENDFILE
SMCR SRT [210,4]PPF654.TES=[210,4]PPF55P.TES/SIZE:80/KEYS:CN1,80
$ 1 DELETE INPUT TO THAT SORT.  CONCATENATE UPDATE FILES.
SMCR PIP
[210,4]PPF55P.TES;*/DE
LP=[210,4]PPF57P.TES
[210,4]PPF57P.TES;*/DE
[210,4]PHASE4.TES=[210,4]PPF53P.TES/RE
[210,4]PHASE4.TES=[210,4]PPF654.TES/AP
[210,4]PHASE4.TES=[210,4]PPF54.TES/AP
[210,4]PPF53P.TES=[210,4]ENDFILE
[210,4]PPF53P.TES;*/DE
[210,4]PPF654.TES;*/DE
[210,4]PPF354.TES;*/DE
$ 1 ALL FILES ARE NOW CLEANED UP AND UPDATES ARE IN 2 FILES,
$ 1 SEPARATED BY PHASE3.TES AND PHASE4.TES.
SMCR PIP
[210,4]PPF353.TES=[210,4]ENDFILE
[210,4]PPF353.TES;*/DE
[210,4]PPF353.TES=[210,4]PHASE4.TES/RE
[210,4]FOR012.DAT=[210,4]ENDFILE
[210,4]ENDR012.DAT;*/DE
[210,4]FOR012.DAT=[210,4]ENDFILE
[210,4]FOR007.DAT=[210,4]ENDFILE
[210,4]FOR007.DAT;*/DE
SMESSAGE TY1 UPDATES WILL NOW BEGIN.
SMESSAGE THIS IS YOUR LAST CHANCE TO STOP TY1 UPDATES (CON OR ABO).
SMCR PIP LP=[210,4]*.TES;*/LI
SMESSAGE/WAIT (IF YOU GO PAST THIS POINT, YOU CANNOT RESTART).
SMCR PIP LP=[210,4]*.TES;*/LI
SMCR PIP [210,4]RM4.COM=[210,4]RM44.COM
SMCR REM RSXBAT
SRUN ASATS.TTK
SMCR PIP [210,4]LQ.TES=*.*/LI
SMESSAGE TY1 UPDATES COMPLETED.
SMCR PIP
[210,4]RM2.POS=[210,4]ENDFILE
[210,4]RM2.POS;*/DE
[210,4]RM4.COM=[210,4]ENDFILE
[210,4]RM4.COM;*/DE
[210,4]RM2.POS=[210,4]FOR012.DAT
[210,4]DUM.ZIP=[210,4]ENDFILE
[210,4]*.ZIP;*/DE
SMESSAGE PREPARE OUTPUT REPORTS FOR TY1.
SMCR PIP POST8.ZIP=ENDFILE
SMCR REM RSXBAT
SRUN POSTP.TTK
SMCR PIP [210,4]CARDS.TES=[210,4]PUNCH.ZIP/RE
SMCR SRT [210,4]PUNCH.ZIP=[210,4]CARDS.TES/SIZE:80/KEYS:CN1,80

```

```

SMCR PIP
[210,4]LQ.TES=[210,4]*.**/LI
[210,4]PUNCH.ZIP=[210,4]DATE.COM/AD
[210,4]POS84.TFS=[210,4]POST8.ZIP
[210,4]POST8.ZIP;*/DE
[210,4]PIN4.TES=[210,4]PUNCH.ZIP
[210,4]PUNCH.ZIP;*/DE
LP=[210,4]*.ZIP)*
[210,4]POST8.ZIP=[210,4]POS84.TES/RE
[210,4]PUNCH.ZIP=[210,4]PIN4.TES/RE
S I START THE OTHER DAILY REPORTS.
SMCR PIP [210,4]UNITS.SAT;20=[210,4]RATCH.SAT
SMCR PIP [210,4]BAT.COM=[210,4]ENDFILE
SMCR PIP [210,4]BAT.COM;*/DE
SMCR PIP [210,4]BAT.COM=[210,4]REP.COM;1/RE
SMCR REM RSXBAT
SRUN RIMS.TTK
SMCR PIP [210,4]LQ.TFS=[210,4]*.**/LI
SMCR PIP [210,4]DUM.SHT=[210,4]ENDFILE
SMCR PIP [210,4]*.SHT;*/DE
SMCR PIP [210,4]BAT.COM;*/DE
SMCR REM RSXBAT
SMCR PIP [210,4]BAT.COM=[210,4]REP.COM;2/RE
SRUN RIMS.TTK
SMCR PIP [210,4]LQ.TFS=*.**/LI
SMCR PIP [210,4]DUM.SHT=[210,4]ENDFILE
SMCR PIP [210,4]*.SHT;*/DE
SMCR PIP [210,4]BAT.COM;*/DE
SMCR PIP [210,4]BAT.COM=[210,4]REP.COM;3/RE
SMCR REM RSXBAT
SRUN RIMS.TTK
SMCR PIP [210,4]LQ.TFS=*.**/LI
SMCR PIP [210,4]DUM.SHT=[210,4]ENDFILE
SMCR PIP [210,4]*.SHT;*/DE
SMCR PIP [210,4]BAT.COM;*/DE
SMCR PIP [210,4]BAT.COM=[210,4]REP.COM;4/RE
SMCR REM RSXBAT
SRUN RIMS.TTK
SMCR PIP [210,4]LQ.TFS=*.**/LI
SMCR PIP [210,4]DUM.SHT=[210,4]ENDFILE
SMCR PIP [210,4]*.SHT;*/DE
SMCR PIP [210,4]BAT.COM;*/DE
SMCR PIP [210,4]BAT.COM=[210,4]REP.COM;5/RE
SMCR REM RSXBAT
SRUN RIMS.TTK
SMCR PIP [210,4]LQ.TFS=*.**/LI
SMCR PIP [210,4]DUM.SHT=[210,4]ENDFILE
SMCR PIP [210,4]*.SHT;*/DE
SMCR PIP [210,4]BAT.COM;*/DE
SMCR PIP [210,4]BAT.COM=[210,4]REP.COM;6/RE
SMCR REM RSXBAT

```

```

$RUN RIMS,TTK
$MCR PIP [210,4]LO.TFS=*.*/*LI
$MCR PIP [210,4]DUM.SHI=[210,4]ENDFILE
$MCR PIP [210,4]*.SHI/*DE
$MCR PIP [210,4]BAT.COM/*DE
$MCR PIP [210,4]BAT.COM=[210,4]RFP.COM;7/RE
$MCR REM RSXBAT
$RUN RIMS,TTK
$MCR PIP [210,4]LO.TES=*.*/*LI
$MCR PIP [210,4]DUM.SHI=[210,4]ENDFILE
$MCR PIP [210,4]*.SHI/*DE
$MCR PIP [210,4]BAT.COM/*DE
$MCR PIP [210,4]UNITS.SAT;20/DE
$MCR PIP [210,4]*.TTK/*DE
$MESSAGE *****
$MESSAGE * END OF ASATS TY1 BATCH UPDATES AND REPORTS *
$MESSAGE * REMEMBER TO: *
$MESSAGE * MAKE CARDS (USE GRDOUT ON THE *
$MESSAGE * FILE [210,4]PUNCH,ZIP) *
$MESSAGE * AND MAKE LABELS (LOAD LABELS INTO *
$MESSAGE * PRINTER AND DO PIP @ [210,4]LA ) *
$MESSAGE * AND SAVE [210,010] ONTO TAPE *
$MESSAGE * (NOTE: YOU ARE STILL IN [5,5]) *
$MESSAGE * (JN1 MT0;TY DATE/UIC=[210,010] *
$MESSAGE * MOU MT0;/OVR *
$MESSAGE * PIP @ [210,10]SANTO *
$MESSAGE * FASTEN THE DIRECTORY TO THE TAPE ) *
$MESSAGE * THIS IS THE END OF THE ASATS TY1 BATCH RUN. *
$MESSAGE *****
$MCR PIP LP1=[210,4]*.TES/*LI
$MESSAGE NOW LOAD 5-PART PAPER INTO THE PRINTER AND
$MESSAGE /WAIT TYPE IN CON (CR) TO PRINT REPORTS.
$MCR PIP LP1=[210,4]*.TES/*LI
$MCR PIP [210,4]*.TES/*DE
$EOJ

```

ORIGINAL PAGE IS
OF POOR QUALITY

3.1.2 TYLORDER.COM, PACKET ORDER LIST

BE
 ZZ1
 RF12,7
 SKPC
 SKPC
 CO1*2
 GC3
 SKCURS1 H
 CO4*5
 DE1
 JTS, LB
 RF12,10
 HD2,1
 ISY
 RF0,12,DATE.COM
 RF13,12

DAILY PACKET ORDER L

HD1,
 HD1, TRANSITION YEAR 1 ITS PC1
 HD1,
 HD1,

HD2, ORD	SEG	LPI	ACQ	REG	ZONE	STR	R	W	CNT	LAST	DEL
TX	REC	CAMS/LPDI	COMMENT								
HD2,	NO	DATE					W	V		CHNG	D

ATE
 RF12,10
 JFS,90
 SC5,46, TOTAL ACQUISITIONS

LALB
 BE
 RF12,7
 SKPC 2
 GC1
 SKCURS1 H

CO2*3
 DE1
 JY3,LR
 RF12,10
 HD2,1

DAILY PACKET ORDER L

ISY
 RF0,12,DATE.COM
 RF13,12
 HD1,
 HD1, TRANSITION YEAR 1 IIS PC2
 HD1,
 HD1,

HD2, ORD	SEG	LPI	ACQ	REG	ZONE	STR	B	W	CNT	LAST	DEL
TX	REC	CAMS/LPDI	COMMENT								
HD2,	NO	DATE					W	V		CHNG	D

ATE
 RF12,10
 JF3,90
 SC3,46, TOTAL ACQUISITIONS
 LALB

DAILY PACKET ORDER L

DAILY PACKET ORDER L

~~37~~
10

BE
 RF12,7
 SKPC 25
 GC1
 SKCURS1 H
 CO2*3
 DE1
 JT3,L8
 RF12,10
 HD2,1
 IST
 RF0,12,DATE.COM
 RF13,12

DAILY PACKET ORDER L

HD1,
 HD1, TRANSITION YEAR 1 US PC25
 HD1,
 HD1,

HD2, ORD	SEG	LPI	ACQ	REG	ZONE	STR	B	W	CNT	LAST	DEL
TX	REC	CAMS/LPDL	COMMENT								
HD2,		NO	DATE				W	V		CHNG	D

ATE
 RF12,10
 JF3,90
 SC3,46. TOTAL ACQUISITIONS
 LALB

BE
 RF12,7
 SKPC 4
 GC1
 SKCURS1 H
 CO2*3

DE1
 JT3,L8
 RF12,10
 HD2,1

DAILY PACKET ORDER L

IST
 RF0,12,DATE.COM
 RF13,12

HD1,
 HD1, TRANSITION YEAR 1 USSR PC4
 HD1,
 HD1,

HD2, ORD	SEG	LPI	ACQ	REG	ZONE	STR	B	W	CNT	LAST	DEL
X	REC	CAMS/LPDL	COMMENT								
HD2,		NO	DATE				W	V		CHNG	D

TE
 RF12,10
 F3,90
 C3,46. TOTAL ACQUISITIONS
 ALB

ORIGINAL PAGE IS
OF POOR QUALITY

BE
RF12,7
SKPC 5
GC1
SKCURS1 H
CO2*3
DE1
JT3,LR
RF12,10
HD2,1
IST
RF0,12,DATE.COM
RF13,12

DAILY PACKET ORDER L

HD1,
HD1, TRANSITION YEAR 1 INDIA PC5
HD1,
HD1,
HD2, ORD SEG LPI ACQ REG ZONE STR B W CNT LAST DEL
TX REC CAMS/LPDL COMMENT
HD2, NO DATE W V CHNG D
ATE

RF12,10
JF3,90
SC3,46, TOTAL ACQUISITIONS
LALB

BE
RF12,7
SKPC 45
GC1
SKCURS1 H
CO2*3
DE1

JT3,LR
RF12,10
HD2,1
IST

DAILY PACKET ORDER L

RF0,12,DATE.COM
RF13,12
HD1,
HD1, TRANSITION YEAR 1 USSR PC45
HD1,
HD1,
HD1,

HD2, ORD SEG LPI ACQ REG ZONE STR B W CNT LAST DEL
TX REC CAMS/LPDL COMMENT
HD2, NO DATE W V CHNG D
ATE

RF12,10
JF3,90
SC3,46, TOTAL ACQUISITIONS
LALB

BE
 RF12,7
 SKPC 7
 GC1
 SKCURS1 H
 CO2*3
 DE1
 JT3,LP
 RF12,10
 HD2,1
 IST
 RF0,12,DATE.COM
 RF13,12
 HD1,
 HD1, TRANSITION YEAR 1 CANADA PC7
 HD1,
 HD1,
 HD2, ORD SEG LPI ACQ REG ZONE STR B W CNT LAST DEL
 TX REC CAMS/LPDL COMMENT
 HD2, NO DATE W V CHNG D
 ATE
 RF12,10
 JF3,90
 SC3,46, TOTAL ACQUISITIONS
 LALB
 BE
 RF12,7
 SKPC 8
 GC1
 SKCURS1 H
 CO2*3
 DE1
 JT3,LP
 RF12,10
 HD2,1
 IST
 RF0,12,DATE.COM
 RF13,12
 HD1,
 HD1, TRANSITION YEAR 1 USSR PC8
 HD1,
 HD1,
 HD2, ORD SEG LPI ACQ REG ZONE STR B W CNT LAST DEL
 TX REC CAMS/LPDL COMMENT
 HD2, NO DATE W V CHNG D
 ATE
 RF12,10
 JF3,90
 SC3,46, TOTAL ACQUISITIONS
 LALB

DAILY PACKET ORDER L

DAILY PACKET ORDER L

BE
RF12,7
SKPC 62
GC1
SKCURS1 H
CO2*3

DE1
JT3, LB
RF12,10
HD2,1

DAILY PACKET ORDER L

IST
RF0,12,DATE.COM
RF13,12

HD1,
HD1, TRANSITION YEAR 1 ARGENTINA PC62

HD1,
HD1,
HD2, ORD SEG LPI ACQ REG ZONE STR B W CNT LAST DEL
TX REC CAMS/LPDL COMMENT
HD2, NO DATE W V CHNG D

ATE
RF12,10
JF3,90
SC3,46, TOTAL ACQUISITIONS

LALB
BE
RF12,7
SKPC 63
GC1
SKCURS1 H
CO2*3

DE1
JT3, LB
RF12,10
HD2,1

DAILY PACKET ORDER L

IST
RF0,12,DATE.COM
RF13,12

HD1,
HD1, TRANSITION YEAR 1 BRAZIL PC63

HD1,
HD1,
HD2, ORD SEG LPI ACQ REG ZONE STR B W CNT LAST DEL
TX REC CAMS/LPDL COMMENT
HD2, NO DATE W V CHNG D

ATE
RF12,10
JF3,90
SC3,46, TOTAL ACQUISITIONS
LALB

3.1.3 RIMS.CMD, RIMS TASK BUILDER COMMANDS

```
DB0: (206,5) RIMS, LP: /SH=RIMS.ODL/MP
MAXBUF=256
ACTFIL=13
UNITS=13
ASG=SY0:11:12:13
ASG=SY0:1:2:3:4
ASG=SY0:5:6:8:9
ASG=TI:7
ASG=LP:10
POOL=60
//
```

3.1.4 RIMS.ODL, RIMS OVERLAY STRUCTURE

```

,ROOT M=*(SEG1-S1,SEG2-S2)
M1 .FCTR MF1=MF2=MF3
MF11 .FCTR SEL=CHAR=COMSTR=DTFINT=INDEX=INPARM=SUBSTP=VERIFY
MF21 .FCTR GET=PUT=ROLL=SETIN1=SETIN2=SETOUT=XXIN1=XXIN2=XXOUT
MF31 .FCTR STATUS=NHITS=ENDSET=LOCATE=LOCREC=LODFMT=LODREC=GETREC
S11 .FCTR UNITS=LAST
S21 .FCTR CIMAIN=CLOSEP=S2F1
S2F11 .FCTR (S2F2,S2F3)
S2F21 .FCTR SEG3-S3,SEG8-SA,SEG13-S13,SEG17-S17
S2F31 .FCTR SEG18-S18,SEG19-S19,SEG20-S20
S31 .FCTR JPRPCR=PRNTID=(SFG4-S4,SEG7-S7)
S41 .FCTR S4F1=(SEG5-S5,SEG6-S6)
S4F11 .FCTR AEINIT=SQZE=CIRP=ADDFN=ADDLT=CFIND
S51 .FCTR AEPR=STAEPR
S61 .FCTR FTFMT=FTCMP
S71 .FCTR TFORMW=EXCMDS=EXCMD=TFORMZ
S81 .FCTR S8F1=(SEG9-S9,SEG10-S10,SEG11-S11,SEG12-S12)
S8F11 .FCTR CISUB1=CFCP=JFDFCP=END=PRNTID=APCOM
S91 .FCTR S9F1=S9F2
S9F11 .FCTR AEINIT=SQZE=CICFDF=CFIND=RLCLPR=RPCLPR=ADDFN=ADDLT
S9F21 .FCTR ADDNM=ADDDT=AEPR=STAEPR
S101 .FCTR APSINT=FTFMT=FTCMP=RLDTRF
S111 .FCTR S11F1-S11F2
S11F11 .FCTR TFORMW=EXCMDS=EXCMD
S11F21 .FCTR APSTUP=TFORMZ=REPR=DISFMT
S121 .FCTR S12F1-S12F2
S12F11 .FCTR APSCNT=AUPST=ADDKEY=APSEL=DELKEY=KOMKEY=LAND
S12F21 .FCTR LNOT=IOR=PART=POST=RANGE=SSORT
S131 .FCTR S13F1=(SEG14-S14,SEG15-S15,SEG16-S16)
S13F11 .FCTR CISUB2=JNSNCR=PRNTJD
S141 .FCTR S14F1-S14F2
S14F11 .FCTR AEINIT=SQZE=CFIND=RLCLPR=ADDFN=ADDLT=ADDNM=ADDDT
S14F21 .FCTR AEPR=STAEPR=FTFMT=FTCMP
S151 .FCTR TFORMW=EXCMDS=EXCMD
S161 .FCTR S16F1-S16F2
S16F11 .FCTR SELECT=IOR=RANGE=COMBIN=LAND=LNOT=HEADER
S16F21 .FCTR STCNT=DELETE
S171 .FCTR S17F1-S17F2
S17F11 .FCTR CISUB3=BEGIN=ATTACH=CLEANP=UNLOCK=PSWRD
S17F21 .FCTR SPLUS=ADDR=SMINUS=DELR
S181 .FCTR S18F1-S18F2-S18F3
S18F11 .FCTR CISUB4=SORTP=CLOSFL=KOMKEY=LAST
S18F21 .FCTR PART=SORTS=SSORT=TJUMP=CMDRI=ADDR=CMPUTE
S18F31 .FCTR LMVTAR=REPR=MOVSEG
S191 .FCTR S19F1-S19F2-S19F3
S19F11 .FCTR CISUB5=EXPAND=FINDD=DISPLA=XREF=SEUNTV=SELREC
S19F21 .FCTR ACCNO=SPCSET=FORMM=ADDR=FORM=DISPDD=REAF=PARSEP
S19F31 .FCTR GETPAR=PARSEC=GETCLD=ADDREC

```

ORIGINAL PAGE IS
OF POOR QUALITY

```

S20: .FCTR S20F1-S20F2-S20F3-S20F4-(SEG21-S21,SEG22-S22)
S20F1: .FCTR CISUR6-APSCNT-APSINT-APSTUP-LMVTAB
S20F2: .FCTR RFPR-TFCRM-DBPRO-AUPOST
S20F3: .FCTR ADDKEY-APSEL-DELKEY-KCMKEY-LAND-LNOT-LOR
S20F4: .FCTR PART-POST-RANGE-SSCHT
S21: .FCTR CAUFIL-AUFILF-XPOST-ADDR
S22: .FCTR CRESTS-DELR-RESTRY-DELRFC-RFPKEY-REPREC
.PSECT APCOM,D,OVR,GBL
.NAME SEG1
.NAME SEG2
.NAME SFG3
.NAME SEG4
.NAME SFG5
.NAME SEG6
.NAME SFG7
.NAME SEG8
.NAME SFG9
.NAME SEG10
.NAME SFG11
.NAME SEG12
.NAME SFG13
.NAME SEG14
.NAME SFG15
.NAME SFG16
.NAME SFG17
.NAME SFG18
.NAME SFG19
.NAME SEG20
.NAME SFG21
.NAME SFG22
.END

```

3.2 NEW AND MODIFIED RIMS PROGRAMS

3.2.1 PROGRAM DESCRIPTIONS

On the pages following are the RIMS programs (subroutines and functions) that were created or modified to provide the requested enhancements.

Name: ADDDT

Purpose: To initialize the Working Buffer Format for a date encountered in the input command.

Linkage:

- Calling sequence: CALL ADDDT (FC, NC, ROW)
- Common blocks used: SY3COM
- Subroutines or functions used: DTEINT
- Files used: None

Input Description: FC = integer variable; character number in array CMD of /SY3COM/ where the date starts (# sign).
NC = integer variable; number of characters in the date literal, including the # sign.

Output Description: ROW = integer variable; the row number of the Working Buffer Format into which the date reference is placed.

Process Description: The pointer to the last used row of WBF, the Working Buffer Format, is incremented by one. \$L is stored into the second word of the row of WBF. The value of 4 is stored into the sixth word of the row of WBF. The value of -1 is stored into the seventh word of the row of WBF. The subroutine DTEINT is used to convert the date from a character string in the command line to a binary integer in the first word of the row of WBF. The row number is stored in ROW, and a return to the calling routine is made.

Name: ADDFN

Purpose: To initialize the Working Buffer Format for a field name encountered in the input command line.

Linkage:

- Calling sequence: CALL ADDFN (FC, NC, ROW)
- Common blocks used: SY3COM
- Subroutines or functions used: COMSTR, SUBSTR
- Files used: None

Input Description: FC = integer variable; character number in array CMD of /SY3COM/ where the field name starts.
NC = integer variable; number of characters in the field name.

Output Description: ROW = integer variable; the row number of the Working Buffer Format, WBF, into which the field name reference is placed.

Process Description: Column two of WBF is searched for the field name. If it is found, the row number is stored into ROW, and a return is made to the calling routine. If it is not found, the pointer to the last used row of WBF is incremented by one, the field name is placed in column two of that row, that row number is stored in ROW, and a return is made to the calling routine.

Name: ADDLT

Purpose: To store a reference to an alphanumeric literal into the Working Buffer Format

Linkage:

- Calling sequence: CALL ADDLT (FC, NC, ROW)
- Common blocks used: SY3COM
- Subroutines or functions used: None
- Files used: None

Input Description: FC = integer variable; character number in array CMD of /SY3COM/ where literal starts (first quote mark).
NC = integer variable; number of characters in the literal, including the beginning and ending quote marks.

Output Description: ROW = integer variable; the row number of the Working Buffer Format into which the reference to the literal is stored.

Process Description: The pointer to the last used row of WBF, the Working Buffer Format, is incremented by one. \$T is stored into the second word of the row of WBF. The value of FC+1 is stored into the fifth word of the row of WBF. The value of NC-2 is stored into the sixth word of the row of WBF. Zero is stored into the seventh word of the row of WBF. The row number is stored into ROW, and a return to the calling routine is made.

Name: ADDNM

Purpose: To initialize the Working Buffer Format for a number literal encountered in the input command.

Linkage:

- Calling sequence: CALL ADDNM (FC, NC, ROW)
- Common blocks used: SY3COM
- Subroutines or functions used: INPARM
- Files used: None

Input Description: FC = integer variable; character number in array CMD of /SY3COM/ where the number starts.
NC = integer variable; number of characters in the number.

Output Description: ROW = integer variable; the row number of the Working Buffer Format, WBF, into which the number reference is placed.

Process Description: The pointer, NWBF, to the last used row of WBF is incremented by one. Then \$L, 4, and -1 are stored in WBF (2, NWBF), WBF (6, NWBF), and WBF (7, NWBF), respectively. INPARM is used to convert the number from a character string in CMD to a binary integer in WBF (1, NWBF). The row number, NWBF, is stored in ROW, and a return is made to the calling routine.

Name: AEINIT

Purpose: To initialize standard areas of core for commands which allow arithmetic expressions in their syntax.

Linkage:

- Calling sequence: CALL AEINIT(IND, SETNO, FMTNO, ERR)
- Common blocks used: SYSCOM, SY3COM, CLTBL
- Subroutines or functions used: SQZE, INPARM
- Files used: Command file (logical unit 13), Message file (logical unit 7)

Input Description: IND = integer variable; indicator to cause special processing for certain commands. Set 0 for SN, JN, and CF commands. Set 1 for DF and JF commands. Set 2 for RP and JP commands.

Output Description: SETNO = integer variable; contains the set number converted from the input command line.
FMTNO = integer variable; contains the format number converted from the input command line, if there is one.
ERR = integer variable; returned zero if no errors found, non-zero if any error is found.

Process Description:

1. Zeroes are stored in all words of /SY3COM/.
2. Blanks are stored in all words of the Working Buffer, WBUF.
3. Word 6 of row 1 of the BY Processing Table, BPT, is initialized to '+\$-&' to cause the first record read in an RP or JP command to create a top level BY change.

4. The first word of array COMMAS in /CLTBL/ is initialized to 1. Variable FC, used to point to the next available character in array CMD of /SY3COM/ (where the input command will be packed), is initialized to 1. The logical unit number for the command file, U13, is retrieved from U(13) in /SYSCOM/. The logical unit number for the message file, U7, is retrieved from U(7).
5. A call is made to SQZE (STR, 1, 80, CMD, FC, NCS, COMMAS) to transfer and compact the input command string (in STR of /SYSCOM/) into array CMD. NCS is returned as the number of characters stored by SQZE into CMD, and pointers to syntactically meaningful commas are returned in COMMAS.
6. FC is incremented by NCS.
7. If $FC \leq 401$, go to step 9.
8. Set ERR = 1 and return to the calling routine.
9. If $COMMAS(1) < 0$, meaning the last input card has been processed for an RP or JP command, then negate COMMAS(1) and go to step 13.
10. If IND = 2, go to step 12.
11. If the last character stored in CMD is a comma, then go to step 13, otherwise increment COMMAS(1) by 1, store FC into COMMAS(COMMAS(1)), and go to step 13.
12. Read, from unit U13, 80 characters into the beginning of array STR. If the actual unit number for the message file = 7, then go to step 5, otherwise echo the input string by writing, to unit U7, 80 characters from the beginning of STR, and go to step 5.

13. Calculate NC, the number of characters in the input set number, = COMMAS(2)-3. Then convert SETNO = INPARM (CMD, 3, NC).
14. If $\text{SETNO} \leq 0$ or $\text{SETNO} \geq \text{TABNO}$, then go to step 8.
15. If $\text{IND} \neq 1$, go to step 18.
16. Calculate NC, the number of characters in the input format number, = COMMAS (3) - COMMAS(2)-1. Then convert FMTNO = INPARM(CMD, COMMAS(2)+1, NC).
17. If $\text{FMTNO} \leq 0$, go to step 8.
18. Return to the calling program.

Name:

AEPR

Purpose:

To parse an arithmetic expression made up of arithmetic operators and operands (Field names, dates, or integer constants), entering operands (or pointers to them) into the working buffer format and building a sequence of internal commands to evaluate the expression and store the value into a specified result variable.

Linkage:

- Calling sequence: CALL AEPR (FC, NC, PTR, ERR)
- Common blocks used: SY3COM
- Subroutines or functions used: ADDFN, COMSTR, DTEINT, INDEX, INPARM, STAEPR, VERIFY
- Files used: None

Input Description:

FC = integer variable; character number in command line at which to begin processing.

NC = integer variable; number of characters to process.

PTR = integer variable; location into which the results are to be stored.

Output Description:

ERR = integer variable; processing or syntax error indicator. Normal command table containing internal commands to evaluate the expression and store the results.

Process Description:

Error flag set to zero and all internal variables set to appropriate value. The expression is scanned for paired

brackets and valid alphanumeric characters. If brackets not paired or any invalid character found ERR set to 2 and return executed. Otherwise, the number of paired brackets saved for later use, character pointers and counters set as needed and the expression is scanned and PASS 1 executed as follows:

1. (a) If the next character encountered is not an open bracket do step 2. Otherwise an open bracket is stored in the next location of VTAB and a -99 is stored in the next location of OPC. (b). If this is the last character the error exit is taken otherwise update pointers and counters and redo step 1(a).
2. If the next character encountered is a closed bracket ERR is set to 2 and return executed. Otherwise do step 3.
3. The next 13 characters are searched for either the end of the scan or an arithmetic operator. If the end of the scan is found the pointer is set to the end of the scan +1 location otherwise the location of the next operator or bracket will be found. Then the current character will be checked to see if it is a pound sign. If it is not a pound sign step 4 is executed. Otherwise the next four characters past the pound sign are checked to see if they are numeric digits. If they are not digits, ERR is set to 2 and return executed. Otherwise DTEINT is called to convert the date to an integer, the appropriate

data is stored in the normal command buffer and step 6 executed.

4. The current character is checked to see if it is a literal value. If it is not a literal value step 5 is executed. Otherwise the literal is converted to an integer, the appropriate data is stored in the normal command buffer and step 6 executed.
5. If none of the above were executed then the next operand is a field name. In this case ADDFN is called to store the data in the normal command buffer and step 6 executed.
- 6(a) If the last operand or operator has been processed a-999 is placed in the current location of OPC and the pass 2 is executed as shown starting at step 8. Otherwise the next operator is checked to see if it is either an open or close bracket. If it is an open bracket the error exit is taken. If it is a closed bracket step 7 is taken. (b) Otherwise the operator is stored in the next location of OPC and step 1(b) executed.
7. The character after the closed bracket is checked. If it is an open bracket the error exit is taken. Otherwise a-88 is stored in the next location of VTAB & OPC and the pointer and counters updated. If the character after the closed bracket was itself a closed bracket step 6 is taken. If the operator after the closed bracket was not a

closed bracket and at the same time located at the end point of the expression to be processed, the error exit is taken otherwise step 6b is taken.

8. The count of data in VTAB & OPC is saved for later use. If the number of paired brackets is zero, PASS 3 is executed as shown starting at step 10. Otherwise step 9 is executed.
9. The OPC table is searched and the innermost paired brackets, as indicated by a -99 and a -88 respectively, is found along with their index location. Then STAEPR is called to store the data into the normal command table. Then the remainder of the VTAB & OPC tables is written over the area where the paired brackets were stored, the count of data in VTAB & OPC is decreased by the amount of data processed by STAEPR, the number of paired brackets is decreased by 1 and step 8 taken.
10. If more than one line of data is left in VTAB & OPC STAEPR is called to store the data into the normal command table. Otherwise the normal command table is updated with the calling argument PTR and column 5 of all used areas of the normal command table is updated to point to the next expression area to be processed.

Name: AUFILE

Purpose: To control the processing for the Add File (AF) and Update File (UF) commands.

Linkage:

- Calling sequence: CALL AUFILE(IJ,IFUN)
- Common blocks used: SYSCOM, SY2COM
- Subroutines or functions used: ADDR, APSCNT, APSINT, APSTUP, AUPOST, GETREC, INPARM, LMVTAB, REPR, SUBSTR, TFORM
- Files used: Unit 7, Logical unit U(11)

Input Description: IJ = integer variable; indicates which of the first five fields of the input format contains the record ID. If IJ=0, the record ID is expected to be in characters two through five and seven through ten of the input record.

IFUN = integer variable; value is one for the AF command and two for the UF command.

Output Description: None

Process Description:

1. Variable TIM1 and U7 are initialized to zero and U(11), respectively.
2. APSINT is called to initialize files U(9) and U(10) for holding new keys and deleted keys.
3. If IFUN=1, row 2 of array BUF is filled with blanks.
4. Read a record into row 1 of array BUF from unit U7. If an end-of-file is read, go to step 20.

5. If $IJ \neq 0$, read the record ID, J, from BUF (via subroutine INPARM) from the location specified in row IJ of page 1 of format array FMT. Go to step 8.
6. Read the upper half of the record ID (via INPARM) from characters 2-5 of row 1 of BUF.
7. If the internal format number (found in FMTID(2)) is not 19, read the lower half of the record ID (via INPARM) from characters 7-10 of row 1 of BUF.
8. If the record ID, J, = 0, go to step 20.
9. If the function indicator, IFUN,=1, go to step 11.
10. Get the record with ID=J from the data base (via GETREC) into row 2 of BUF. If the status returned from GETREC was zero (implying that the record was present and its internal format was already loaded in page 2 of array FMT), then go to step 12. If the status was two (implying that the record was present, but its format had to be loaded), then go to the next step, otherwise go to step 3.
11. Set TIM1=1 and call LMVTAB to load the move table in array MOVTAB.
12. If TIM1=0, then go to step 11. If IFUN=1, then go to step 14.

13. Each field number in row 2 of the move table is passed to subroutine APSTUP which, if the field is a key field, builds an entry on logical unit U(10) to show a key field value that needs to be deleted.
14. Call subroutine TFORM to transfer data from fields of the record in row 1 of BUF into fields of the record in row 2 of BUF, as directed by the move table.
15. If IFUN=1, set the first word of row 2 of BUF equal to the length of the record in that row.
16. Each field number in row 2 of the move table is passed to APSTUP which, if the field is a key field, builds an entry on logical unit U(9) to show a key field value that needs to be added.
17. If IFUN=2, subroutine REPR is called to replace the existing record with the new one just created from the old one and the new fields.
18. If IFUN=1, subroutine ADDR is called to add the record just read to the data base.
19. Go back to step 3.
20. Write 'SP' on unit 7, use APSCNT and AUPOST to delete old keys and post new keys, and then return to the calling routine.

Name: BLDTBF

Purpose: To convert a data base format into a form suitable for standardized processing.

Linkage:

- Calling sequence: CALL BLDTBF(P)
- Common blocks used: SY2COM, SY3COM
- Subroutines or functions used: COMSTR
- Files used: None

Input Description: P = integer variable; page number of array FMT in /SY2COM/ where the data base format is currently stored. .

Output Description: None

Process Description:

1. Initialize K to zero.
2. Initialize I to two, the first row of the data base format which contains a field name.
3. If NWBF, the pointer to the last used row of WBF, the Working Buffer Format in /SY3COM/, is = 0, go to step 8.
4. Initialize J to one, the first row of WBF.
5. If WBF (2, J) = \$R, \$T, or \$L, go to step 7.
6. Use COMSTR to compare the field name in column 2 of WBF to the field name in column 2 of FMT at row I. If a match is found, go to step 9.
7. Increment J by one. If $J \leq NWBF$, go back to step 5.

8. Increment NWBF by one. Transfer columns 2, 4, and 5 of row I of FMT to columns 2, 4, and 5 of row J of WBF.
9. Increment NTBF, the pointer to the last used row of TBF, the Target Buffer Format in |SY3CØM|, by one. Store a one in TBF (1,NTBF) and store J in TBF (2,NTBF). Transfer columns 3, 4, and 5 of row I of FMT to columns 3, 4, and 5 of row NTBF of TBF.
10. If the target field type, TBF (7,NTBF),=4, set k=1.
11. Initialize I to one. Search column 2, row I, of WBF for \$R, \$T, or \$L. If any one is found, go to step 15.
12. Search column 2 of WBF for the field names "UNLOAD" and "LSD". If "UNLOAD" is found, go to step 13. If "LSD" is found, go to step 14. If neither is found, go to step 15.
13. Set TBF(4,1) = -I. If TBF(4,2) ≠0, return to the calling routine, otherwise go to step 15.
14. Set TBF(4,2) = -I. If TBF(4,1) ≠0, return to the calling routine.
15. If TBF(4,1) = 0, meaning "UNLOAD" was not found, increment NWBF by one, store "UNLOAD" as a field

name in column 2 of WBF at row NWBF,
set TBF(4,1) = -NWBF, and then, if
TBF(4,2) \neq 0, return to the calling
routine.

- 16.. Increment NWBF by one, store "LSD"
as a field name in column 2 of WBF at
row NWBF, set TBF(4,2) = -NWBF, and
return to the calling routine.

Name: CFCR

Purpose: To direct the overall processing sequence for the CF command.

Linkage:

- Calling sequence: CALL CFCR
- Common blocks used: SYSCOM, SY2COM, SY3COM, CLTBL
- Subroutines or functions used: AEINIT, CICFDF, APSINT, SETIN1 XXIN1, FTFMT, FTCMP, GETREC, TFORMW, EXCMDS, APSTUP, TFORMZ, REPR, APSCNT, AUPOST
- Files used: Message file (logical unit 7), Deleted keys file (logical unit 10), New keys file (logical unit 9)

Input Description: None

Output Description: None

Process Description:

1. Initialize file pointer U7 to the value stored in U(7) of /SYSCOM/ and use AEINIT to initialize /SY3COM/, returning input set number in SETNO and a non-zero in ERR if any errors were found.
2. If ERR = 0, go to step 4.
3. Write on U7, "Command terminated due to syntax error." and return to the calling routine.
4. If COMMAS(1) \leq 2, meaning there are no replacement clauses, then go to step 3.
5. Use CICFDF(0, ERR) to complete the command interpretation, returning ERR non-zero if any errors were found.
6. If ERR \neq 0, go to step 3.
7. Use APSINT to initialize for storing deleted keys on file U(10) and new

- keys on file U(9). Use SETIN1 to initialize for returning record ID's via XXIN1. Use XXIN1 (RID) to return the first record ID in RID.
8. If RID = 0, return to the calling routine.
 9. Use FTFMT (RID, 1, ERR) to complete the buffer formats with information about fields whose names occurred in the input command, returning ERR non-zero if an error occurred.
 10. If ERR \neq 0, go to step 3.
 11. Use FTCMP to complete starting location information in the Working Buffer Format, WBF in /SY3COM/.
 12. Loop through the Target Buffer Format, TBF in /SY3COM/, comparing its column 2 contents with the values found in column 2 of the Multilevel Move Table, MLMT in /SY3COM/. When a match is found at row I of TBF and row K of MLMT, extract L = column 1 of row K of MLMT, so that L points to the matching row of the Source Buffer Format, SBF in /SY3COM/. Then transfer the key field indicator from SBF(1,L) to TBF(3,I), the data base format row number from SBF(2,L) to TBF(4,I), the starting character position in the data base record from SBF(5,L) to TBF(5,I), the length of the field from SBF(6,L) to TBF(6,I), and the type of the field from SBF(7,L) to TBF(7,I).
 13. Use GETREC(1, RID, STAT) to retrieve record RID into row 1 of BUF, the record buffer in /SY2COM/, returning STAT non-zero if there was any problem with the retrieval.

14. If $STAT \neq 0$, bypass this record by going to step 23.
15. Use TFORMW(1,1) to transfer needed fields from row 1 of BUF to WBUF, directed by row 1 of the Move Table Control Table, MTCT in /SY3COM/.
16. Set EF = 0 and use EXCMDs to execute the commands in the Normal Command Table, CTBL in /SY3COM/, returning CFLAG as false if there was a failure of a relational clause, and returning EF non-zero if a command could not be executed for some reason.
17. If CFLAG is false, bypass this record by going to step 23.
18. If $EF \neq 0$, bypass this record by going to step 23.
19. Loop through TBF looking for key fields. When one is found ($TBF(3,I) \neq 0$), then use APSTUP to store the key to be deleted on file U(10).
20. Use TFORMZ(1,1) to transfer changed fields from WBUF to row 1 of BUF.
21. Loop through TBF, and for each key field found, use APSTUP to store the new key on file U(9).
22. Use REPR to replace the old record in the data base with the revised one in row 1 of BUF.
23. Use XXINI(RID) to get the next record ID into RID.
24. If $RID \neq 0$, meaning there was a next record, then go back to step 13.
25. Use APSCNT to retrieve the number of keys to be changed in the data base.
26. Use AUPQST to delete the keys stored on file U(10).

27. Use AUPOST to add the keys stored on file U(9).
28. Return to the calling routine.

Name: CFIND

Purpose: To locate any single character of one string within another string.

Linkage:

- Calling sequence: CALL CFIND (STRA, STA, NCA, STRB, STB, NCB, LOCA, LOCB)
- Common blocks used: None
- Subroutines or functions used: INDEX
- Files used: None

Input Description: STRA = integer array name; start of string to be searched
STA = integer variable; character number of STRA at which to begin search
NCA = integer variable; number of characters in STRA to be searched
STRB = integer array name; start of string containing characters for which to search
STB = integer variable; character number of STRB where search characters start
NCB = integer variable; number of search characters in STRB to be used.

Output Description: LOCA = integer variable; character number in STRA where the first find was made. Zero if no characters in STRB were found in STRA.
LOCB = integer variable; character number in STRB of the character found. Zero if none found.

Process Description: Successive characters of STRA, starting at STA and continuing for NCA characters, are individually compared to the characters in STRB (via the INDEX function) until a match is found or STRA is exhausted.

<u>Name:</u>	CHAR
<u>Modification Purpose:</u>	To allow conversion of negative numbers.
<u>Linkage Modification:</u>	<ul style="list-style-type: none"> ● Calling sequence: No change ● Common blocks used: No change ● Subroutines or functions used: The Fortran function IABS is now used. ● Files used: No change
<u>Input Description Modification:</u>	V = integer variable; may now contain values less than zero.
<u>Output Description Modification:</u>	STR = integer array name; leftmost position will contain the minus character if the input value in V was negative.
<u>Process Description Modification:</u>	Convert the absolute value of V to a character string by the original process. Then, if V is negative, store the minus character as the leftmost character of the output field.

Name: CICFDF

Purpose: To direct the command interpretation phase of the CF, DF, and JF commands.

Linkage:

- Calling sequence: CALL CICFDF(IND, ERR)
- Common blocks used: SY3COM, CLTBL
- Subroutines or functions used: CFIND, RPCLPR, RLCLPR
- Files used: None

Input Description: IND = integer variable; indicator for which command is being processed. Set zero for CF command. Set non-zero for DF or JF commands.

Output Description: ERR = integer variable; error indicator. Returned zero if no errors found. Returned non-zero if any error is found.

Process Description:

1. The error indicator, ERR, is initialized to 0. The replacement expression found indicator, REF, is initialized to 0. The commas pointer, CP, is initialized to 2.
2. If IND \neq 0, CP is changed to equal 3.
3. The first character pointer, FC, is calculated = COMMAS(CP)+1. The number of characters, NC, between commas is calculated = COMMAS(CP+1)-FC.
4. Use CFIND to check the character at FC for a single quote mark, a number sign, or a numeric character. If any of these are found, go to step 9.
5. Use CFIND to check all NC characters for an arithmetic operator, parenthesis, equal sign or period. If one of these is found, go to step 7.

6. Set $ERR = 1$ and return to the calling routine.
7. If an equal sign was found, go to step 14.
8. Since no equal sign was found, the clause must be a relational clause. Since all relational clauses are to be before replacement clauses, check REF to see if a replacement clause has been found. If $REF \neq 0$, go to step 6.
9. Use RLCLPR(FC, NC, ERR) to process the relational clause and return ERR non-zero if errors were found.
10. If $ERR \neq 0$, go to step 6.
11. Increment CP by 1. If $CP \neq \text{COMMAS}(1)$, go to step 3.
12. If $IND \neq 0$, return to the calling routine.
13. The CF command must have a replacement clause, so if $REF = 0$, go to step 6, otherwise return to the calling routine.
14. Set $REF = 1$ and use RPCLPR (FC, NC, I, IND, ERR) to process the replacement clause. I was returned previously from CFIND as the location of the equal sign, and ERR will be returned non-zero from RPCLPR if any errors are found.
15. If $ERR \neq 0$, then go to step 6, otherwise go to step 11.

Name: CMAIN

Purpose: To direct control to the proper command routine.

Linkage:

- Calling Sequence: CALL CMAIN(String)
- Common blocks used: SYSCOM, SECCOM
- Subroutines or functions used:
CISUB1, CISUB2, CISUB3, CISUB4,
CISUB5, CISUB6, CLOSEP, JPRPCR,
MODE, STATUS, SUBSTR, VERIFY
- Files used: None

Input Description: STRING = alphanumeric character string containing the command line syntax.

Output Description: None

Processing Description:

1. Set SECURE(17)=1
2. Search command line for blank character position, if found continue with next step, else return to calling routine.
3. Store command line in STR, and process next step.
4. Compare command in CMD with each element of the command table [SWITCH(K)], if CMD = SWITCH(K), go to the next step, else write 'INVALID COMMAND' message, then go to step 20.

5. Verify first command issued is a BE command. If SECURE(K) \neq 1, go to step 20, else continue with the next step.
6. If $k < 6$, go to step 12; if $k > 9$, go to step 7, else call CISUB1(k), then go to step 19.
7. If $k > 16$, go to the next step, else CALL CISUB2(k), then go to step 19.
8. If $k > 19$, go to the next step, else CALL CISUB3(k), then go to step 19.
9. If $k > 24$, go to the next step, else CALL CISUB4(k), then go to step 19.
10. If $k > 37$, go to the next step, else CALL CISUB5(k), then go to step 19.
11. Call CISUB6(K,FLAG), then go to step 18.
12. If $k \neq 1$, go to the next step, else go to step 19.
13. If $k \neq 2$, go to the next step, else CALL STATUS(0), then go to step 19.
14. If $k \neq 3$, go to next step, else CALL MODE, then go to step 19.
15. If $k \neq 4$, go to the next step else set CIND=0, then go to step 17.
16. Set CIND=1.
17. Use JPRPCR(CIND) to process the RP, or JP commands, indicated from step 15 or 16, then go to step 19.

18. If FLAG \leq 1, go to next step, else
CALL SUBSTR, which stores STR into
STRING. Return to step 1.
19. Call CLOSEP, process next step.
20. Return to calling routine.

~~343~~

Name: CIRP

Purpose: To direct the activities of parsing the command line, building tables, and building buffer formats for the RP and JP commands.

Linkage:

- Calling sequence: CALL CIRP(CIND, RECID, ERR)
- Common blocks used: SY3COM, CLTBL
- Subroutines or functions used: COMSTR, CFIND, ADDFN, AEPR, ADDLT, INDEX, INPARM, FTFMT, FTCMP
- Files used: None

Input Description: CIND = integer variable; command indicator. Set 0 for RP command or non-zero for JP command.
RECID = integer variable; contains the record ID (accession number) of the first record of the input set.

Output Description: ERR = integer variable; error indicator. Returned zero if no errors are found and non-zero if any error is found.

Process Description:

1. Counters and pointers are initialized.
2. If there are some clauses in the command, go to step 4.
3. Set ERR = 1 and return to the calling routine.
4. If there are no characters in the clause, go to step 3.
5. If this clause is the first one and it is not a BY clause, then go to step 3.
6. If this clause is not a BY clause and not the first clause, then go to step 16.

7. If this BY clause is occurring after an E&E BY clause has occurred, then go to step 3.
8. If this is the sixth BY clause, then go to step 3.
9. If there is no grouping field name for this BY clause, then go to step 3.
10. If this is an E&E BY clause, then if there are no report expressions, then go to step 3, otherwise store zeroes in the first 3 columns of the BY Processing Table, BPT in /SY3COM/, for this BY clause and go to step 15.
11. Use CFIND to check the grouping field name for arithmetic operators or parentheses. If any are found, go to step 13.
12. Set columns 2 & 3 of BPT = 0 for this BY clause, use ADDFN to store the grouping field name in the Working Buffer Format, WBF in /SY3COM/, store the row number returned by ADDFN into column 1 of BPT for this BY clause, and go to step 15.
13. Store the negative of the next available row number of WBF in column 1 of BPT for this BY clause. Store \$R in column 2 of that row of WBF. Store the next available row number of the Normal Command Table, CTBL in /SY3COM/, into column 2 of BPT for this BY clause. Store 4 and -1 into columns 4 & 5 of the \$R row of WBF.
14. Use AEPR to generate the commands which evaluate the arithmetic expression. If AEPR found any syntax errors, go to step 3. Calculate the number

- of commands generated by AEPR and store this number into column 3 of BPT for this BY clause.
15. Move to the next pair of commas. If there is none, go to step 29, otherwise go back to step 4.
 16. If this report expression is not a text type, go to step 18.
 17. Use ADDLT to create an entry in WBF for this text literal and store values in the Target Buffer Format, TBF in /SY3COM/, to cause this text to be printed at the start or conclusion of this BY clause (depending on whether single or double quote mark characters were used). Go to step 15.
 18. Compare the beginning characters of the report expression with an internal table of function names. If no match is found, go to step 21.
 19. If this function reference is found in an E&E BY clause, then go to step 3.
 20. Use ADDFN to create the field name reference in WBF for the field name specified in the function. Create a \$R row in WBF for the results of the function to be carried and create a reference in TBF to get the results printed upon the conclusion of this BY clause. Based on which function was specified, store an initialization value in column 1 of the \$R row of WBF. Then go to step 15.
 21. Use INDEX to check for an equal sign in the report expression. If there is one, go to step 24.
 22. This report expression is only a field name. If it has greater than 12 characters, go to step 3.

23. Use ADDFN to create a reference to the field name in WBF, and create an entry in TBF to cause the value of this field to be printed at the beginning of this BY clause. Then go to step 15.
24. Use CFIND to determine if this report expression begins with an I or a D. If it begins with neither, then go to step 3.
25. Use INPARM to convert the input field width to a binary integer. If it is greater than 99, then go to step 3.
26. Create a results field (\$R) in WBF and a target field for printing in TBF. Store the target field type in TBF as a 1 or a 2 based on whether the report expression began with an I or a D, respectively.
27. Initialize column 4 of BPT for this BY clause if it has not already been done.
28. Use AEPR to process the arithmetic expression to the right of the equal sign, and accumulate the number of commands generated into column 5 of BPT for this BY clause. Then go to step 15.
29. If CIND = 0, then set NLVLS, the number of data base levels, = 1, otherwise set NLVLS = 2.
30. Use FTFMT (RECID, NLVLS, ERR) to search the data base formats, collecting field information for WBF and SBF, the Source Buffer Format in /SY3COM/. If ERR is returned non-zero, go to step 3.
31. Now that the type and length of data base fields are known, this information is used to complete needed portions of WBF

and TBF where just field names and functions with field names are the report expressions.

32. Where function results are called for in WBF, a command is entered in the Function Command Table, FCTBL in /SY3COM/.
33. Use FTCMP to generate starting character positions in WBF and initial values in WBUF.
34. Use FTCMP to generate starting character positions with two spaces between fields in TBF.
35. Return to the calling routine.

<u>Name:</u>	CISUB1
<u>Purpose:</u>	<ul style="list-style-type: none"> ● To direct selection of the appropriate subroutine(s) for execution of the associated command.
<u>Linkage:</u>	<ul style="list-style-type: none"> ● Calling sequence: Call CISUB1(k) ● Common blocks used: none ● Subroutines or functions used: CFCR, JFDFCR, END ● Files used: none
<u>Input Description:</u>	<p>k = integer variable; indicator for which command is being processed.</p> <p>Set k=6 for CF command. Set k=7 for DF command. Set k=8 for JF command. Set k=9 for EN command.</p>
<u>Output Description:</u>	None
<u>Processing Description:</u>	<ol style="list-style-type: none"> 1. Set LK = k-5 2. If LK≠1, go to step 3, else use CFCR to process the CF command, then go to step 9. 3. If LK≠2 go to step 5, else process the next step. 4. Set CIND = 0, indicates the DF command is being processed, go to step 7. 5. If LK≠3 go to step 8, else go to the next step. 6. Set CIND = 1, indicates the JF command is being processed, go to next step.

7. Use JFDFCR to process the DF or JF command indicated from step 4 or 6. Go to step 9.
8. If LK \neq 4 go to step 9 else use END to process the EN command.
9. Return to calling routine.

Name: CISUB2

Purpose: To direct selection of the appropriate subroutine(s) for execution of the associated command.

Linkage:

- Calling sequence: Call CISUB2(k)
- Common blocks used: none
- Subroutines or functions used:
JNSNCR, SELECT, COMBIN, HEADER,
STCNT, DELETE
- Files used: none

Input Description: k = integer variable; indicator for which command is being processed. Set k=10 for SN command. Set k=11 for JN command. Set k=12 for SK command. Set k=13 for CO command. Set k=14 for HD command. Set k=15 for SC command. Set k=16 for DE command.

Output Description: None

Processing Description:

1. Set LK= k-9.
2. If LK≠1, go to step 3, else set CIND=0, go to step 4.
3. If LK≠2, go to step 5, else set CIND=1, go to next step.
4. Use JNSNCR to process the SN or JN command, then go to step 10.
5. If LK≠3, go to step 6, else use SELECT to process the SK command. Then go to step 10.

6. If LK \neq 4, go to step 7 else use COMBIN to process the CO command. Then go to step 10.
7. If LK \neq 5, go to step 8, else use HEADER to process the HD command. Then go to step 10.
8. If LK \neq 6, go to step 9, else use STCNT to process the SC command. Then go to step 10.
9. If LK \neq 7, go to step 10, else use DELETE to process the DE command.
10. Return to calling routine.

Name: CISUB3

Purpose: To direct selection of the appropriate subroutine(s) for execution of the associated command.

Linkage:

- Calling Sequence: Call CISUB3(k)
- Common blocks used: SECCOM
- Subroutines or functions used:
BEGIN, CLEANP, UNLOCK, SPLUS,
SMINUS
- Files used: None

Input Description: k= integer variable, indicator for which command is being processed. Set k=17, for BE command. Set k=18, for S+ command. Set k=19, for S- command.

Output Description: None

Processing Description:

1. Set LK= k-16.
2. If LK≠1, go to step 3, else use BEGIN, CLEANP; and UNLOCK to process the command line. Then go to step 5.
3. If LK≠2, go to step 4, else use SPLUS to process the S+ command. Then go to step 5.
4. If LK≠3, go to step 5 else use SMINUS to process the S- command.
5. Return to calling routine.

Name: CISUB4 .

Purpose: To direct selection of the appropriate subroutine(s) for execution of the associated command.

Linkage:

- Calling Sequence: Call CISUB4(k)
- Common blocks used: None
- Subroutines or Functions: SORTP, TJUMP, CMDRI, MOVSEG
- Files used: None

Input Description: k= integer variable; indicator for command being processed. Set k=20, for SO command. Set k=22, for JT command. Set k=21, for JS command. Set k=22, for JT command. Set k=23, for CM command. Set k=24, for MO command.

Output Description: None

Processing Description:

1. Set LK= k-19
2. If LK≠1, go to step 3, else set CIND=1, go to step 4.
3. If LK≠2, go to step 5, else set CIND=2, go to next step.
4. If CIND=1 process SORTP.
If CIND=2 process SORTP, go to step 8,
5. If LK≠3 go to step 6, else use TJUMP, then go to step 8.
6. If LK≠4, go to step 7, else use CMDRI to process the CM command.
Then go to step 8.

7. If LK~~≠~~5, go to step 8, else use
MOVSEG to process the MO command.
8. Return to calling routine.

Name: CISUB5

Purpose: To direct selection of the appropriate subroutine(s) for execution of the associated command.

Linkage:

- Calling Sequence: Call CISUB5(k)
- Common blocks used: None
- Subroutines or functions used:
EXPAND, DISPLA, XREF, SEUNIV,
SELREC, SPCSET, FORMM, DISPDD,
REAF, PARSEP, PARSEC, ADDRREC

Input Description:

k= integer variable, indicator for command being processed.

Set k=25, for EX command.

Set k=26, for PA command.

Set k=27, for DI command.

Set k=28, for XR command.

Set k=29, for SU command.

Set k=30, for SR command.

Set k=31, for SS command.

Set k=32, for DD command.

Set k=33, for FO command.

Set k=34, for RF command.

Set k=35, for GP command.

Set k=36, for GC command.

Set k=37, for AR command.

Output Description: None

Processing Description

1. Set LK= k-24.
2. If LK≠1, go to step 3, else set CIND=0, go to step 4.

3. If LK \neq 2, go to step 5, else set CIND=1, go to next step.
4. Use EXPAND to process the EX or PA command, indicated from step 2 or 3. Then go to step 16.
5. If LK \neq 3, go to step 6, else use DISPLA, to process the DI command. Then go to step 16.
6. If LK \neq 4, go to step 7, else use XREF to process the XR command. Then go to step 16.
7. If LK \neq 5, go to step 8, 3lse use SEUNIV to process the SU command. Then go to step 16.
8. If LK \neq 6, go to step 9, else use SELREC to process the SR command. Then go to step 16.
9. If LK \neq 7, go to step 10, else use SPCSET to process the SS command. Then go to step 16.
10. If LK \neq 8, go to step 11, else use FORMM to process the DD command. Then go to step 16.
11. If LK \neq 9, go to step 12, else use DISPDD to process the FO command. Then go to step 16.
12. If LK \neq 10, go to step 13, else use REAF to process the RF command. Then go to step 16.
13. If LK \neq 11, go to step 14 else use PARSEP to process the GP command. Then go to step 16.

14. If $LK \neq 12$, go to step 15 else use PARSEC to process the GC command. Then go to step 16.
15. If $LK \neq 13$, go to the next step, else use ADDREC to process the AR command.
16. Return to calling routine.

Name: CISUB6

Purpose: To direct selection of the appropriate subroutine(s) for execution of the associated command.

Linkage:

- Calling sequence: Call CISUB6(k, FLAG)
- Common blocks used: None
- Subroutines or functions used: CAUFIL, XPOST, ADDKEY, DBPRO, DELKEY, CRESTS, DELREC, REPKEY, REPREC.
- Files used: None

Input Description:

k= integer variable, indicator for which command is being processed.

Set k=38, for AF command.

Set k=39, for UF command.

Set k=40, for UP command.

Set k=41, for PO command.

Set k=42, for VP command.

Set k=43, for AK command.

Set k=44, for DK command.

Set k=45, for RE command.

Set k=46, for KY command.

Set k=47, for NK command.

Set k=48, for DS command.

Set k=49, for DR command.

Set k=50, for RK command.

Set k=51, for RR command.

Output Description:

FLAG = integer variable; where
FLAG = 1 means do not allow data base
to be altered.
FLAG = 0 means do allow data base to
be altered.
FLAG = 2 means process input line as
next command.

Processing Description:

1. Set LK= k-37, and set FLAG=0, go
to next step.
2. If LK≠1, go to step 3, else set
CIND=1, go to step 4.
3. If LK≠2, go to step 5, else set
CIND=2, go to next step.
4. Use CAUFIL to process the AF or
UF command, indicated from step
2 or 3. Then go to step 17.
5. If LK≠3, 4, or 5 go to step 6,
else use XPOST to process the
UP, PO, or VP command. Then
go to step 17.
6. If LK≠6, go to step 7, else use
ADDKEY to process the AK command.
Then go to step 17.
7. If LK≠7, go to step 8, else use
DBPRO(FLAG) to pick up variable
FLAG. If FLAG≠0 go to step 17,
else use DELKEY to process the
DK command. Then go go step 17.
8. If LK≠8, go to step 9, else set
CIND = 1 and go to step 13.

9. If LK \neq 9, go to step 10, else set CIND=2, go to step 13.
10. If LK \neq 10, go to step 11, else set CIND=3, go to step 12.
11. If LK \neq 11, go to step 14, else set CIND=4, go to next step.
12. Use DBPRO(FLAG) to pick up variable, FLAG. If FLAG \neq 0 go to step 17, else next step.
13. Use CRESTS(CIND) to process the RE, KY, NK, or DS command. Then go to step 17.
14. If LK \neq 12, go to step 15, else use DBPRO(FLAG) to determine FLAG status. If FLAG \neq 0, go to step 17, else use DELREC to process the DR command. Then go to step 17.
15. If LK \neq 13, go to step 16, else use REPKEY to process the RK command. Then go to step 17.
16. If LK \neq 14, go to step 17, else use REPREC to process the RR command.
17. Return to calling routine.

Name: CRESTS

Purpose: To parse the command line for the RE, KY, NK, and DS commands.

Linkage:

- Calling sequence: CALL CRESTS (IFN)
- Common blocks used: SYSCOM, SY2COM
- Subroutines or functions used: CLOSEP, INDEX, INPARM, LODFMT, RESTRX
- Files used: None

Input Description:

IFN = integer variable, indicator for command being processed.
 If IFN=1, process RE command.
 If IFN=2, process KY command.
 If IFN=3, process NK command.
 If IFN=4, process DS command.

Output Description: None

Processing Description:

1. Search the command line for comma, if a comma is found a non-zero value is assigned to variable k which indicates the numeric location of the comma in the command line. If a comma is not found, variable k is set to zero.
2. If k=0 go to step 9, else continue with the next step.
3. M = integer variable representing the set number within the command line. If m=0 go to step 8, else go to the next step.

4. If IFN=4, go to step 9, otherwise verify the existence of a format number, convert to integer, and store the converted integer into FMTID(2).
5. Call LODFMT, process next step.
6. Use RESTRX(M, IFN) to process the appropriate command, with (M) containing the set number and (IFN) the proper command switch.
7. Call CLOSEP, continue with next step.
8. Return to calling routine.
9. If IFN≠4; this implies the DS command was attempted, however a syntax error exists in the command line, therefore go to step 8. If IFN=4, k = integer value of first blank. This value is used to determine the number of characters to convert from alpha to integer. Return to step 3.

Name:

DBPRO

Purpose:

To prevent accidental alteration of the data base for certain commands.

Linkage:

- Calling Sequence: Call DBPRO (FLAG)
- Common blocks used: SYSCOM
- Subroutines or functions used: COMSTR, SUBSTR, VERIFY
- Files used: Logical units U(7) and U(13).

Input Description:

A command line containing the input command line plus a "Yes," "Y," "No," or "N" after the command or a command line containing the input line then another input line containing a "yes" or "No" in response to an output query.

Output Description:

FLAG = integer variable; where FLAG = 1 means do not allow data base to be altered and FLAG=0 means do allow data base to be altered.

FLAG=2 means the current command is ignored, but process array STR as next command.

Process Description:

1. Initialize FLAG to zero.
2. Use VERIFY to verify the existence of a non-blank character at or beyond position 60 of the command line. Variable k is set.
3. If k=0 go to step 6. If k≠0, use COMSTR to compare the character at k of STR with a 'Y'.

4. If COMSTR=0, return to the calling routine.
5. Use COMSTR to compare the character at k of STR with a 'N'. If COMSTR=0, go to step 14.
6. Write the command line to the message file with a 'YES or NO?' query.
7. Read the response to the query from the command file.
8. Use VERIFY to check for a non-blank character in response to the query.
9. If k=0, go to step 14.
10. Use COMSTR to compare the response to the query with 'YES'. If COMSTR=0, return.
11. Use COMSTR to compare the response to the query with 'NO'. If COMSTR=0, go to step 14.
12. Set FLAG=2.
13. Use SUBSTR to move the response data into STR, and to return to the calling routine.
14. Set FLAG=1.
15. Return to the calling routine.

<u>Name:</u>	DISFMT
<u>Purpose:</u>	To direct output to appropriate file unit.
<u>Linkage:</u>	<ul style="list-style-type: none"> ● Calling sequence: CALL DISFMT (ID,L,STR) ● Common blocks used: SYSCOM ● Subroutines or functions used: None ● Files used: U(12), U(8)
<u>Input Description:</u>	<p>ID = integer variable, indicator for file unit assignment. If ID=1 assign U7=U(12) If ID≠1 assign U7=U(8) L = integer variable, indicates length of input string STR. STR = alphanumeric character string.</p>
<u>Output Description:</u>	STR = alphanumeric character string.
<u>Processing Description:</u>	<ol style="list-style-type: none"> 1. If ID=1, set U7=U (12) and process the next step. 2. If ID≠1, set U7=U(8) and process the next step. 3. LL=L, stores input length into LL, go to the next step. 4. If LL>30, set LL=30.

5. If $U7 \neq 7$, write the contents of STR with a format length of 30 words to unit = U7.
6. If $U7 = 7$, write the contents of STR with a format length of 15 words, process next step.
7. Return to calling routine.

Name:

DTEINT

Purpose:

To convert date format to/from a binary integer.

Linkage:

- Calling sequence:
CALL DTEINT (FUNC, INT, STR, ST, NC)
- Common blocks used: None
- Subroutines or functions used:
SUBSTR, CHAR, INPARM, MOD
- Files used: None

Input Description:

FUNC = Indicator, if FUNC = 0, converts character string (STR) to an integer (INT). If FUNC \neq 0, converts an integer to a character string (STR).
INT = Integer input.
STR = character string input.
ST = starting position of character string.
NC = number of characters of STR to be converted.

Output Description:

INT = Integer output from converted character string.
STR = character output from input integer.

Process Description:

In addition to the input variables, this routine contains an internal Julian day conversion table DTAB. DTAB is a one-dimensional array with each element representing the total number of days from the base year to year 'N', where N is the relative position of the array element representing an offset from the base year. If FUNC indicates an integer is to be converted to a character string the input integer date is tested for an invalid date. If this date is greater than the

greatest value of DTAB, the input date is replaced with that particular DTAB element and the conversion process continued. However, if the integer date is less than or equal to zero, blanks are moved to the output string (STR). Assuming the integer date is greater than zero, DTAB is searched until a value that is greater than or equal to the input integer is found. The input integer minus the previous table value gives the day segment of the Julian date. The year segment is then calculated by adding the base year to the DTAB index minus a constant of two. Having converted the integer date to a Julian date format, the results are then converted to an alphanumeric character string by use of the CHAR subroutine. To convert from an alpha Julian date format the year and day segments are calculated. The year portion is subtracted from the base year to serve as an index to pick up the appropriate DTAB element. Once this element is obtained this value is added to the day segment to produce the output integer.

Name: EXCMD

Purpose: To perform the operations specified in one row of a command table.

Linkage:

- Calling sequence: CALL EXCMD (TBL, ROW, ERR, CFLAG)
- Common blocks used: SY3COM
- Subroutines or functions used: SUBSTR, DTEINT, INPARM, CHAR, COMSTR
- Files used: None

Input Description: TBL = integer array name; starting location of the table containing the command to be executed. ROW = integer variable; contains the row number of TBL where the command to be executed is stored.

Output Description: ERR = integer variable; contains zero on normal command execution or non-zero when command cannot be executed.
CFLAG = logical variable; contains .TRUE. normally, but is set to .FALSE. when the command is a logical comparison and the comparison fails.

Process Description: Refer to the Command Table and Command Operations Table layouts as a supplement to this description. The value retrieved from the references in columns 1-4 of the command table will be referred to as OPND(1), OP, OPND(2), and RESULT, respectively, in this description.

1. ERR and CFLAG are initialized to zero and true, OP is retrieved from TBL (2, ROW), OPND(1)'s pointer, P, is retrieved from TBL (1, ROW), and N is initialized to one.

2. If $OP > 0$ and < 17 , go to step 4.
3. Set $ERR = 1$ and return to calling routine.
4. If $OP > 10$, go to step 24.
5. If $P = 0$, go to step 3.
6. If $P < 0$, negate P , retrieve $OPND(N)$ from $REG(P)$, an array in $/SY3COM/$, and go to step 10.
7. Retrieve $OPND(N)$'s type from column 5 at row P of the Working Buffer Format, WBF , in $/SY3COM/$.
8. If $OPND(N)$'s type = 0, then if $OP < 5$, then go to step 3, otherwise if $N = 2$, then go to step 3, otherwise go to step 20.
9. Convert the value of $OPND(N)$ from the Working Buffer, $WBUF$ in $/SY3COM/$, based on the type, using either subprogram $SUBSTR$ (type < 0, a binary integer), $INPARM$ (type = 1, a numeric character string), or $DTEINT$ (type = 2, a date character string).
10. If $N = 1$, then set $N = 2$, retrieve a new P from $TBL(3, ROW)$, and go to step 5.
11. If $OP > 4$, go to step 22.
12. Perform the arithmetic operation specified by OP , using $OPND(1)$ and $OPND(2)$ and storing the result in $RESULT$. If $OPND(2)$ of a divide operation = 0, then go to step 3.
13. Retrieve a new P for $RESULT$ from $TBL(4, ROW)$.
14. If $P = 0$, go to step 3.
15. If $P < 0$, negate P , store $RESULT$ in $REG(P)$, and go to step 19.
16. Retrieve $RESULT$'s type from column 5 at row P of WBF .
17. If type = 0, go to step 3.
18. Convert the value in $RESULT$ into $WBUF$ based on the type, and using subprogram $SUBSTR$ (type < 0), $CHAR$ (type = 1), or $DTEINT$ (type = 2).

19. Return to the calling routine.
20. If OPND(2)'s pointer in TBL (3, ROW) = 0, or if OPND(2)'s type \neq 0, then go to step 3.
21. Perform an alphanumeric comparison between OPND(1) and OPND(2) and set I to be negative, zero, or positive according to whether OPND(1) < OPND(2), OPND(1) = OPND(2), or OPND(1) > OPND(2), respectively. Go to step 23.
22. Perform arithmetic comparison by setting I = OPND(1) - OPND(2).
23. Leave CFLAG = true or change CFLAG = false based on the following table and then return to calling routine:

	I < 0	I = 0	I > 0
OP = 5	true	false	false
OP = 6	true	true	false
OP = 7	false	true	false
OP = 8	true	false	true
OP = 9	false	true	true
OP = 10	false	false	true
24. If OP = 16, go to step 40.
25. Retrieve OPND(2)'s pointer, P2, from TBL(3, ROW). If P2 = 0, go to step 3.
26. If OP > 13, go to step 35.
27. If OPND(2) in WBUF is blanks, go to step 39.
28. Retrieve OPND(1) from WBUF using SUBSTR.
29. If OP = 11, set OPND(2) = OPND(1) + 1, and go to step 34.
30. Retrieve OPND(2) from WBUF, converting based on its type and using INPARM or DTEINT.
31. Perform a numeric comparison between OPND(1) and OPND(2). If OPND(1) = OPND(2), go to step 39.

32. If OPND(1) > OPND(2), then if OP = 13, then go to step 39, otherwise go to step 34.
33. If OP = 12, then go to step 39.
34. Use SUBSTR to store OPND(2) into OPND(1)'s place in WBUF and go to step 39.
35. Perform an alphanumeric comparison between OPND(1) and OPND(2). If OPND(1) = OPND(2), to to step 39.
36. If OPND(1) > OPND(2), then if OP = 13, then go to step 39, otherwise go to step 38.
37. If OP = 14, go to step 39.
38. Use SUBSTR to store OPND(2)'s character string in WBUF into OPND(1)'s character string in WBUF.
39. Return to calling routine.
40. If P = 0, go to step 3.
41. If P < 0, negate P, retrieve RESULT from REG(P), and go to step 13.
42. Retrieve OPND(1)'s type from column 5 at row P of WBF. If type = 0, go to step 44.
43. Convert OPND(1) from WBUF into RESULT based on type using subprogram SUBSTR (type < 0), INPARM (type = 1), or DTEINT (type = 2). Go to step 13.
44. Retrieve RESULT's pointer, P2, from TBL (4, ROW). If P2 = 0, go to step 3.
45. Transfer OPND(1)'s character string in WBUF to RESULT's location in WBUF, using SUBSTR, and then to go step 39.

Name: EXCMDS

Purpose: To execute a sequence of related command rows in a command table.

Linkage:

- Calling sequence: CALL EXCMDS (TBL, SR, NR, ERRFNC, CFLAG)
- Common blocks used: SY3COM
- Subroutines or functions used: EXCMD, SUBSTR
- Files used: None

Input Description:

TBL = integer array name; starting location of the table which contains the commands to be executed.

SR = integer variable; starting row number within the command table.

NR = integer variable; number of rows to be executed.

ERRFNC = integer variable; indicator for what procedure is to be followed if an error occurs: Zero means do nothing to the results field; non-zero means store blanks or zero in the results field(depending on field type).

Output Description:

ERRFNC = integer variable; set to zero if no errors were encountered. Set to one if an error was encountered.

CFLAG = logical variable; contains .TRUE. except when a relational comparison command has failed, then it contains .FALSE..

Process Description:

1. The last row to be processed is calculated into LR, ERRFNC is saved in EF and set = 0, and I is initialized to SR.
2. A call is made to subroutine EXCMD to execute the command at row I.

3. If CFLAG from EXCMD is returned with a value of false, return immediately to the calling routine.
4. If the error indicator from EXCMD is returned non-zero, go to step 7.
5. Increment I by 1.
6. If $I > LR$, return to the calling routine.
7. Set ERRFNC = 1 and retrieve P from column 5 of the current row of the command table. P is the row number to which a jump should be made.
8. If EF = 0, go to step 13.
9. Retrieve the result pointer, P1, from column 4 of row P-1 of the command table.
10. If P1 = 0, go to step 13.
11. Retrieve the result type from column 5 of row P1 of the Working Buffer Format.
12. Based on type, store binary zeroes ($\text{type} < 0$), alpha zeroes ($\text{type} > 0$), or blanks ($\text{type} = 0$) into the result location in the Working Buffer.
13. Set $I = P$ and go to step 6.

Name:

FØRM

Purpose:

To store a new format definition into the data base.

Linkage:

- Calling sequence: CALL FØRM(DBRID,LEN)
- Common blocks used: SYSCØM, SY2CØM
- Subroutines or functions used: ADDR, CLØSEP, CØMSTR
- Files used: Logical unit U(11)

Input Description:

DBRID = integer variable; data base record ID. This is the format number.
LEN = integer variable; length (in integer *4 words) of the record which the new format describes.

Output Description:

None

Process Description:

1. The maximum number of fields is initialized to 27, the actual number of fields is initialized to 0, the internal buffer pointer is initialized for the first field description, and the logical unit for reading the input data is initialized to U(11).
2. A loop is executed (up to the maximum number of fields allowed) which reads the ID number, name, start character, length, type, and key code for each field, storing the information into the internal buffer, BUF, and incrementing the buffer pointer and actual field counter for each read. If the beginning three characters of any field name are "END," the loop is terminated.

3. The actual number of words used in the internal buffer is calculated and stored in BUF(1). BUF(2) is set to 0. BUF(4) is set to the input variable LEN, and BUF(8) is set to the actual number of fields read from the input data.
4. ADDR is called to store a record of length BUF(1), beginning at BUF(2), into data base record DBRID. CLØSEP is called to flush internal I/Ø page buffers.
5. FMTID(1) and FMTID(2) in /SY2CØM/ are set to 0 to cause any subsequent command that uses a format to go to the data base to retrieve the current definition of that format.
6. Control is returned to the calling routine.

Name: FTCMP

Purpose: To calculate starting character positions for fields in generated formats.

Linkage:

- Calling sequence: CALL FTCMP (A, NS)
- Common blocks used: SY3COM
- Subroutines or functions used: SUBSTR
- Files used: None

Input Description: A = integer array name; starting location of the array which contains the format to be completed.
NS = integer variable; the number of spaces to be inserted between fields.

Output Description: A = integer array name; starting location of the array which contains the completed format.

Process Description:

1. The start character counter, SC, is initialized to one, as is the row counter, ROW.
2. The length of the field at ROW is transferred from A(6, ROW) to L.
3. If L = 0, then processing is finished, so return to the calling routine.
4. If A(2, ROW) contains \$T then bypass start character calculations for this row (since text remains in the command line instead of being transferred to the Working Buffer, WBUF in /SY3COM/) and go to step 7.
5. If A(2, ROW) contains \$L or \$R, then use SUBSTR to initialize WBUF from A(1, ROW).
6. Store SC into A(5, ROW) and calculate the next SC = SC+L+NS.
7. Increment ROW by one and go to step 2.

Name: FTFMT

Purpose: To retrieve information from formats associated with records in the same family tree.

Linkage:

- Calling sequence: CALL FTFMT (RECID, NLVLS, ERR)
- Common blocks used: SYSCOM, SY2COM, SY3COM
- Subroutines or functions used: LOCREC, GET, INPARM, LODFMT, COMSTR, PRNTID
- Files used: Message file

Input Description: RECID = integer variable; record ID of a record at the lowest level of the data base where format information collection is to begin.
NLVLS = integer variable; the number of levels of the data base to be used in tracing the family tree for format information.

Output Description: ERR = integer variable; returned non-zero if any errors were encountered.

Process Description:

1. The input record ID, RECID, is moved to variable ID. U7 is initialized to whatever unit has been designated as the message file, and the data base top level indicator, TLF, is set to zero.
2. The pointer, NMTCT, to the last used row of the Move Table Control Table, MTCT in /SY3COM/, is incremented by one.
3. The next available row number of the Multilevel Move Table, MLMT in /SY3COM/, is stored in column one of row NMTCT of MTCT.
4. Subroutine LOCREC is used to locate the pointer to the record with accession number = ID. If the pointer is found, go to step 7.

5. Write "Record not in data base" on unit U7.
6. Set $ERR = 1$ and return to calling routine.
7. Use GET to retrieve the pointer to the record, and use GET again to retrieve the format number of the record from the second word of the record (characters 5-8). Convert the format number character string to a binary integer via INPARM, and store the format number in FMTID(1) of /SY2COM/ and in column 3 of row NMTCT of MTCT.
8. Use LODFMT to retrieve the format whose number is in FMTID(1) and store the format in page 1 of FMT in /SY2COM/. If the format was not found in the data base, go to step 6.
9. Calculate the last used row, N, of page 1 of FMT as $FMT(6, 1, 1) + 1$.
10. For each field name in column 2 of the Working Buffer Format, WBF in /SY3COM/, which is not \$R, \$T, or \$L, compare that name against the names up to row N in column 2 of page 1 of FMT, and if a match is found, do steps 11-21, otherwise just move to the next name in WBF until they are all processed, and then go to step 22.
11. If the row number, I, of WBF where the match was found, $= -TBF(4,1)$, then negate $TBF(4,1)$ and go to step 13. TBF is the Target Buffer Format in /SY3COM/, and $TBF(4,1)$ contains the negative of the row number of the field whose name is "UNLOAD". This value was placed in $TBF(4,1)$ by subroutine BLDTBF when a target field type of 4 was encountered, meaning a special output conversion was desired which depended on the contents of the "UNLOAD" field.
12. If $I = -TBF(4,2)$, then negate $TBF(4,2)$. This is the row number of the "LSD"

- field which is similar to the "UNLOAD" field in step 11.
13. The length of the field is transferred from column 4 of FMT to WBF(6,I).
 14. The type of the field is transferred from column 5 of FMT to WBF(7,I).
 15. The pointer, NSBF, to the last used row of the Source Buffer Format, SBF of /SY3COM/, is incremented by one.
 16. The key field indicator is transferred from column 6 of FMT to SBF(1, NSBF).
 17. The starting character number is transferred from column 3 of FMT to SBF(5, NSBF).
 18. The length of the field is transferred from column 4 of FMT to SBF(6, NSBF).
 19. The type of field is transferred from column 5 of FMT to SBF(7, NSBF).
 20. The row number within FMT of the field is stored in SBF(2, NSBF) for later use in the Change Field command.
 21. The pointer, NMLMT, to the last used row of MLMT is incremented by one. The value of NSBF is stored in MLMT (1, NMLMT), and the value of I is stored in MLMT (2, NMLMT).
 22. After processing all fields of WBF that were found in the format for records at this data base level, calculate the number of rows of MLMT which were generated $(=NMLMT - MTCT(1, NMTCT) + 1)$ and store it in MTCT(2, NMTCT).
 23. Check column 4 of all fields of WBF. If any lengths are still = 0, then more formats need to be examined if possible, so go to step 24, otherwise set TBF(4,1) and TBF(4,2) to zero if they are still negative and return to the calling routine.
 24. If NMTCT \neq NLVLS, go to step 26.

25. Write "Unidentified field(s)" on unit U7, and go to step 6.
26. If TLF \neq 0, then go to step 25, otherwise use PRNTID to get the next level record ID and go to step 2.

Name: JFDFCR

Purpose: To direct the overall processing sequence for the JF and DF commands.

Linkage:

- Calling sequence: CALL JFDFCR(CIND)
- Common blocks used: SYSCOM, SY2COM, SY3COM, CLTBL
- Subroutines or functions used: AEINIT, LODFMT, CICFDF, BLDTBF, SETIN1, XXIN1, FTFMT, FTCMP, GETREC, TFORMW, PRNTID, EXCMDS, TFORMZ, DISFMT.
- Files used: Message file (logical unit 7)

Input Description: CIND = integer variable; command indicator. Set zero for DF and non-zero for JF.

Output Description: None

Process Description:

1. Initialize file pointer U7 to the value stored in U(7) of /SYSCOM/.
2. Use AEINIT to initialize /SY3COM/, returning the input set number in SETNO, the input format number in FMTNO, and error indication of non-zero in ERR.
3. If ERR = 0, go to step 5.
4. Write on U7, "Command terminated due to syntax error.", and return to the calling routine.
5. Store FMTNO in FMTID(2) and use LODFMT (2,HIT) to load the format into page 2 of array FMT in /SY2COM/, returning HIT as zero if the format could not be found in the data base.
6. If HIT = 0, then write on U7, "Format not found. ", and return to the calling routine.

7. If COMMAS(1) = 3, meaning there were no clauses in the input command, then go to step 10.
 8. Use CICFDF to process the clauses in the input command, returning ERR non-zero if any errors were found.
 9. If ERR \neq 0, go to step 4.
 10. Use BLDTBF(2) to convert the data base format in page 2 of FMT to a standard format in the Target Buffer Format, TBF in /SY3COM/.
 11. Use SETIN1 to initialize the input set for record ID's to be returned by XXIN1. Use XXIN1(RID) to return the first record ID in RID.
 12. If RID = 0, meaning there were no records in the input set, return to the calling routine.
 13. Set NL, the number of levels in the data base to be used, to 1 or 2, depending on whether CIND = 0 or \neq 0, respectively. Then use FTFMT(RID, NL, ERR) to complete the buffer formats with information about fields whose names occurred in the input command, returning ERR non-zero if an error occurred.
 14. If ERR \neq 0, then go to step 4.
 15. Use FTCMP to complete starting location information in the Working Buffer Format, WBF in /SY3COM/.
- Fill row 2 of the target buffer array, BUF, with blanks.
16. Initialize I, the pointer to the desired row of MTCT, the Move Table Control Table in /SY3COM/, to 1.
 17. Transfer the format number for records at this data base level from MTCT(3,I) to FMTID(1). This prevents an unnecessary retrieval of the format record by GETREC.

18. Use GETREC(1, RID, STAT) to retrieve record RID into row 1 of BUF, the record buffer in /SY2COM/, returning STAT non-zero if there was any problem with the retrieval.
19. If STAT \neq 0, ignore this record by going to step 27.
20. Use TFORMW(1,I) to transfer data from row 1 of BUF to WBUF, the Working Buffer in /SY3COM/, as directed by row I of MTCT.
21. If $I \geq$ NMTCT, the last used row of MTCT, then go to step 23.
22. Increment I by 1, use PRNTID to get record RID's parent record, PID, set RID to PID, and go back to step 17.
23. Set EF = 1, CFLAG=.TRUE., and use EXCMDS to execute the commands in CTBL, the Normal Command Table in /SY3COM/, returning CFLAG as false if any of the relational clauses failed to be true.
24. If CFLAG is false, go to step 27.
25. Use TFORMZ(2,1) to transfer data from WBUF to row 2 of BUF.
26. Use DISFMT to display the record in row 2 of BUF according to the format in FMTID(2).
27. Use XXIN1 (RID) to retrieve the next record ID in RID.
28. If RID \neq 0, meaning there was a next record, go back to step 16.
29. Return to the calling routine.

Name: JNSNCR

Purpose: To direct the overall processing sequence for the JN and SN commands.

Linkage:

- Calling sequence: CALL JNSNCR(CIND)
- Common blocks used: SYSCOM, SY2COM, SY3COM, CLTBL
- Subroutines or functions used: AEINIT, RLCLPR, SETIN1, SETOUT, XXIN1, FTFMT, FTCMP, GETREC, TFORMW, PRNTID, EXCMDS, XXOUT, ENDSET
- Files used: Message file (logical unit 7), pointer lists file (logical unit 5)

Input Description: CIND = integer variable; command indicator. Set zero for SN and non-zero for JN.

Output Description: None

Process Description:

1. Initialize file pointer U7 to the value stored in U(7) of /SYSCOM/. Initialize the comma array pointer, CP, to 2. Initialize the number of records selected, HITS, to 0.
2. Use AEINIT to initialize /SY3COM/, returning the input set number in SETNO, and returning ERR non-zero if any errors were found.
3. If ERR = 0, go to step 5.
4. Write on U7, "Command terminated due to syntax error.", and return to the calling routine.
5. If COMMAS(1) < 3, meaning that there were no relational clauses input, go to step 4.

6. Calculate FC, the first character of the relational clause, = COMMAS(CP)+1. Calculate NC, the number of characters in the relational clause, = COMMAS(CP+1)-FC. Then use RLCLPR(FC,NC,ERR) to process the relational clause, building buffer formats and commands to be executed, and returning ERR non-zero if any errors were found.
7. If ERR \neq 0, go to step 4.
8. Increment CP by 1. If CP < COMMAS(1), go back to step 6.
9. Use SETIN1 to initialize the input set for record ID's to be returned by XXIN1. Use SETQUT to initialize file U(5) to receive selected record ID's.
10. Use XXIN1 (RID) to return the first record ID in RID.
11. If RID = 0, meaning there were no records in the input set, go to step 27.
12. Set NL, the number of levels of the data base to be used, to 1 or 2, depending on whether CIND = 0 or \neq 0, respectively. Then use FTFMT(RID, NL, ERR) to complete the buffer formats with information about fields whose names occurred in the input command, returning ERR non-zero if an error occurred.
13. If ERR \neq 0, go to step 4.
14. Use FTCMP to complete starting location information in the Working Buffer Format, WBF in /SY3COM/.
15. Initialize I, pointer to the desired row of MTCT, the Move Table Control Table in /SY3COM/, to 1. Set R = RID.
16. Transfer the format number for records at this data base level from MTCT (3,I) to FMTID(1). This prevents an unnecessary retrieval of the format record by GETREC.

17. Use GETREC(1, R, STAT) to retrieve record R into row 1 of BUF, the record buffer in /SY2COM/, returning STAT non-zero if there was any problem with the retrieval.
18. If STAT \neq 0, ignore this record by going to step 25.
19. Use TFORMW(1,I) to transfer data from row 1 of BUF to WBUF, the Working Buffer in /SY3COM/, as directed by row I of MTCT.
20. If $I \geq$ NMTCT, the last used row of MTCT, then go to step 22.
21. Use PRNTID to get record R's parent record, PR. Then set R = PR, increment I by 1, and go back to step 16.
22. Set EF = 1 and use EXCMDS to execute the commands in CTBL, the Normal Command Table in /SY3COM/, returning CFLAG as false if any of the relational clauses failed to be true.
23. If CFLAG is false, go to step 25.
24. Increment HITS by 1, and use XXOUT (RID) to store the selected record ID on file U(5).
25. Use XXIN1(RID) to return the next record ID in RID.
26. If RID \neq 0, meaning there is a next record, go back to step 15.
27. Use ENDSET (HITS, U(5)) to create and display an entry in the status table of sets.
28. Return to the calling routine.

Name: JPRPCR

Purpose: To direct the overall processing sequence for the JP and RP commands.

Linkage:

- Calling sequence: CALL JPRPCR(CIND)
- Common blocks used: SYSCOM, SY2COM, SY3COM
- Subroutines or functions used: AEINIT, SETIN1, XXIN1, CIRP, GETREC, TFORMW, PRNTID, EXCMDS, COMSTR, SUBSTR, TFORMZ
- Files used: Message file (logical unit 7), Report file (logical unit 12).

Input Description: CIND = integer variable; command indicator. Set zero for RP command and non-zero for JP command.

Output Description: None

Process Description:

1. Initialize file pointers U7 and U12 to the values stored in U(7) and U(12) of /SYSCOM/. Initialize first and last record indicator, FLREC, to zero.
2. Use AEINIT to initialize values in common, return the input set number in SETNO, and return an error indicator, ERR, non-zero if any errors were found.
3. If ERR = 0, go to step 6.
4. Write on U7, "Command terminated due to syntax error."
5. Return to the calling routine.
6. Use SETIN1 to initialize set number SETNO for returning record ID's via XXIN1.
7. Use XXIN1 to return the first record ID, RECID, from the input set.

8. If RECID = 0, then write on U7, "Null Input Set." and go to step 5.
9. Use CIRP(CIND, RECID, ERR) to interpret the command, build tables and buffer formats, and return ERR non-zero if any errors were found.
10. If ERR \neq 0, go to step 4.
11. Initialize the Move Table Control Table pointer, MTCTP, to one.
12. Transfer the format number for this level of the data base from MTCT(3, MTCTP) to FMTID(1). This prevents actual retrieval of the format record by GETREC, since it is not needed.
13. Use GETREC(1, RECID, STAT) to get record RECID into row 1 of BUF in /SY2COM/, returning STAT non-zero if a problem occurred.
14. If STAT \neq 0, ignore this RECID by going to step 28.
15. Use TFORMW(1, MTCTP) to transfer data from row 1 of BUF to WBUF, the Working Buffer in /SY3COM/, based on the directions provided by row MTCTP of MTCT.
16. If MTCTP \geq last used row of MTCT, NMTCT, then go to step 18.
17. Use PRNTID to get the record ID, PID, of the parent of RECID. Store PID into RECID, increment MTCTP by one, and go back to step 12.
18. Initialize the BY Processing Table pointer, BPTP, to one.
19. Get the Grouping Field Name pointer, GFN, from column 1 of row BPTP of the BY Processing Table, BPT of /SY3COM/.
20. If GFN = 0, go to step 33.
21. If GFN > 0, go to step 24.

22. Negate GFN, set ERR = 0, and use EXCMDS to execute the commands in CTBL, the Normal Command Table of /SY3COM/, as specified by columns 2 and 3 of row BPTP of BPT, returning ERR non-zero if the commands could not be executed for some reason.
23. If $ERR \neq 0$, assume no change in this GFN, and go to step 25.
24. Use COMSTR to compare the new GFN in WBUF to the current GFN in column 6 of BPT. If they are different, go to step 49.
25. If $BPTP \geq$ last used row of BPT, NBPT, then go to step 27.
26. Increment BPTP by 1 and go back to step 19.
27. Set ERR = 0 and use EXCMDS to execute all the commands stored in the Function Command Table, FCTBL in /SY3COM/.
28. Use XXIN1 to get the next record ID into RECID.
29. If $RECID \neq 0$, meaning there is another record to be processed, then go back to step 11.
30. Set FLREC = 2 to mean that the last record is being processed.
31. Set BPTP = 1 and get GFN from column 1 of row 1 of BPT.
32. If $GFN < 0$, negate GFN.
33. If FLREC = 0, meaning we are processing the first record, then set FLREC = 1 and go to step 39.
34. Initialize the local BPT pointer, LBPTP, to the current value of BPTP.
35. If $BPT(1, LBPTP) = 0$, go to step 38.
36. Transfer all function results and concluding text from this BY level by setting $PFLAG = 2 * LBPTP$ and

- calling TFORMZ(2, PFLAG) to transfer from WBUF to row 2 of BUF.
37. If LBPTP < last used row of BPT, NBPT, then increment LBPTP by 1 and go back to step 35.
 38. Write to U12 the first 120 characters of row 2 of BUF. If FLREC = 2, meaning we were processing the last record, then return to the calling routine.
 39. Blank out the first 120 characters of row 2 of BUF.
 40. Initialize LBPTP = BPTP.
 41. Set ERR = 1 and use EXCMD5 to execute the commands of CTBL specified by columns 4 and 5 of row LBPTP of BPT.
 42. Transfer from WBUF to row 2 of BUF field values, calculations, and beginning text by setting PFLAG = 2 * LBPTP - 1 and calling TFORMZ(2, PFLAG).
 43. If LBPTP \geq NBPT, then go to step 27.
 44. Increment LBPTP by 1 and get GFN from BPT(1, LBPTP).
 45. If GFN = 0, go to step 41.
 46. If GFN > 0, go to step 48.
 47. Negate GFN, set ERR = 0, and use EXCMD5 to execute the commands of CTBL specified by columns 2 and 3 of row LBPTP of BPT. If ERR is returned non-zero, go to step 41.
 48. Use SUBSTR to store the new value of GFN from WBUF to column 6 of row LBPTP of BPT, and then go to step 41.
 49. Use SUBSTR to store the new value of GFN from WBUF to column 6 of row BPTP of BPT, and then go to step 33.

C-2

Name: PRNTID

Purpose: To return the record ID of the next higher level record in the same family tree of an inverted tree logically structured data base.

Linkage:

- Calling sequence: CALL PRNTID (CID, PID, TLFLAG)
- Common blocks used: None
- Subroutines or functions used: None
- Files used: None

Input Description: CID = integer variable; child record ID.

Output Description: PID = integer variable; parent record ID.
TLFLAG = integer variable; set zero or non-zero depending on whether output parent ID is not or is at the top level of the data base, respectively.

Process Description: Coded specifically for ASATS, the child record ID consists of the segment number concatenated with the acquisition date. To get the parent record ID, the acquisition date portion (lower 16 bits) is set to zero. TLFLAG is set to 1 since ASATS parent records are at the top level of the data base.

Name: RLCLPR.

Purpose: To parse a relational clause of the form AE.OP.AE (where AE is an arithmetic expression, and OP is a comparison operator) and build a table of commands to evaluate the clause.

Linkage:

- Calling sequence: ;CALL RLCLPR (FC, NC, ERR)
- Common blocks used: SY3COM
- Subroutines or functions used: COMSTR, INDEX, ADDLT, CFIND, AEPR, ADDNM, ADDDT, ADDFN
- Files used: None

Input Description: FC = integer variable; first character number of the string to be processed in array CMD of /SY3COM/.
NC = integer variable; number of characters in the string to be processed.

Output Description: ERR = integer variable; returned zero if no errors are found, non-zero if an error is found.

Process Description:

1. Initialize ERR = 0, F = FC, N = NC, and K = 0.
2. If the character at F is not a single quote mark, go to step 11.
3. Use INDEX to find the next quote mark at J.
4. If $J \neq 0$, go to step 6.
5. Set ERR = 1 and return to the calling routine.
6. If $J \leq F+1$, go to step 5.
7. Use ADDLT (F, J-F+1, V(1)) to add the literal to the Working Buffer Format, WBF in /SY3COM/, getting the row number of WBF returned in V(1).
8. Set K = 1 to indicate that the left hand side of the relational clause has been processed.
9. Recalculate the number of characters remaining, $N, = N - (J - F + 1)$.

10. Reset the first character pointer, F,
= J+1.
11. Use INDEX to find the first period in N
characters beginning at F and store
the location in I.
12. If I = 0, go to step 5.
13. Use COMSTR to compare the four characters
that start at I with an internal table of
legal operators. If a match is found, then
J is set to the row number of the internal
table, otherwise go to step 5.
14. The actual operator number, OP, to eventually
be stored in the command table is calculated
by adding 4 to J.
15. J is initialized to 1.
16. If K = 0, go to step 19.
17. If I \neq F, go to step 5.
18. J is reset to 2, F is incremented by 4 to
set it past the operator, N is decremented
by 4 to account for the operator characters,
and a jump to step 20 is made.
19. N is set to the number of characters to the
left of the operator by setting it equal to
I-F.
20. If $N \leq 0$, go to step 5.
21. If the character at F is not a single quote
mark, go to step 28.
22. If $N \leq 2$, go to step 5.
23. If the character at F+N-1 is not a single
quote mark, go to step 5.
24. Use ADDLT (F, N, V(J)) to add the literal to
WBF and receive the row number in V(J).
25. If J = 2, go to step 27.
26. Set J = 2, F = I+4, and N = NC-N-4 to adjust
to the right hand side of the operator, and
go back to step 20.
27. Increment NCTBL, the pointer to the last
used row of the Normal Command Table, CTBL in

- /SY3COM/, by 1. Store V(1) in CTBL
(1, NCTBL), OP in CTBL(2, NCTBL), V(2) in
CTBL (3, NCTBL), NCTBL+1 in CTBL (5, NCTBL),
and return to the calling routine.
28. Use CFIND to locate any arithmetic operator,
storing its location in K.
29. If K = 0, go to step 33.
30. Increment NWBF, the pointer to the last used
row of WBF, by 1. Store NWBF in V(J), 0 in
WBF(1,NWBF), \$R in WBF(2,NWBF), 4 in
WBF (6, NWBF), and -1 in WBF (7, NWBF).
31. Use AEPR (F, N, V(J), ERR) to process the
arithmetic expression, building commands in
CTBL which store a result at V(J) of WBF,
and returning ERR non-zero if any errors were
found.
32. If ERR \neq 0, go to step 5, otherwise go to
step 25.
33. Use CFIND to determine if the character at
F is the number sign (K will be returned = 1
and L will be returned = 8) or a numeric
character (K will be returned = 1 and L will
be >8).
34. If K = 0 (implying a field name), go to step
38.
35. If L \neq 8 (implying a numeric literal), use
ADDNM (F, N, V(J)) to add the number to
WBF, receiving the row number back in V(J),
and go to step 25.
36. If N \neq 5, go to step 5.
37. Use ADDDT (F, N, V(J)) to add the date to WBF,
receiving the row number in V(J), and go
to step 25.
38. If N > 12, go to step 5.
39. Use ADDFN (F, N, V(J)) to add the field name
to WBF if necessary, receiving the row number
back in V(J), and go to step 25.

Name: RPCLPR

Purpose: To parse a replacement clause of the form FN = AE (where FN is a field name and AE is an arithmetic expression) and build a table of commands to perform the replacement.

Linkage:

- Calling sequence: CALL RPCLPR (FCS, TNC, LOCEQL, IND, ERR)
- Common blocks used: SY3COM
- Subroutines or functions used: INDEX, ADDFN, COMSTR, ADDLT, CFIND, ADDDT, AEPR, ADDNM
- Files used: None

Input Description:

FCS = integer variable; first character number of the string to be processed in array CMD of /SY3COM/.

TNC = integer variable; total number of characters in the string to be processed.

LOCEQL = integer variable; the character number which is the location of the equal sign in the input string. If zero, the equal sign will be searched for internally.

IND = integer variable; command indicator to allow special processing for different commands. A value of zero means the Change Field command is being processed. A non-zero value means the Display Formatted or Joint Display Formatted command is being processed.

Output Description: ERR = integer variable; set to zero if no errors are found and set to non-zero if an error is found.

Process Description:

1. If $LOCEQL \leq 0$, then INDEX is used to find the equal sign and its character position is stored in I, otherwise I is set = LOCEQL.
2. If the equal sign is located, go to step 4.

3. Set ERR = 1 and return to the calling routine.
4. Initialize the first character pointer, FC, to FCS.
5. Calculate the number of characters, NC, in the field name = I-FC.
6. If $NC \leq 0$, go to step 3.
7. If $NC > 12$, go to step 3.
8. Use ADDFN (FC, NC, K) to add the field name to the Working Buffer Format, WBF in /SY3COM/, if it is not already there, and receive back the row number of WBF in K.
9. If IND = 0, then increment NTBF, the pointer to the last used row of TBF, the Target Buffer Format in /SY3COM/, by one, store a one in TBF (1, NTBF) and store K in TBF (2, NTBF).
10. Move the first character pointer, FC, to the first character past the equal sign by setting $FC = I+1$. Calculate NC, the number of characters to the right of the equal sign, by $NC = TNC-I$.
11. If $NC \leq 0$, go to step 3.
12. If the character at FC is not a single quote mark, go to step 17.
13. If $NC < 3$, go to step 3.
14. Use ADDLT (FC, NC, L) to add the text literal to WBF and receive the row number back in L.
15. Increment NCTBL, the pointer to the last used row in CTBL, the Normal Command Table in /SY3COM/, by one. Store L in CTBL (1, NCTBL), 16 in CTBL (2, NCTBL), K in CTBL (4, NCTBL), and NCTBL+1 in CTBL (5, NCTBL).
16. Return to the calling routine.
17. Use CFIND to determine if the character string to the right of the equal sign is an arithmetic expression by locating any +, -, *, /, (, or), and pointing to it with I.

18. If $I \neq 0$, then call AEPR (FC, NC, K, ERR) to process the arithmetic expression, creating commands in CTBL to evaluate the expression and store the result in K, returning ERR as non-zero if any errors were found, otherwise go to step 21.
19. If $ERR \neq 0$, go to step 3.
20. Return to the calling routine.
21. Check the character at FC for the number sign or a numeric character via CFIND.
22. If neither was found, the right side of the equal is assumed to be a field name. If $NC > 12$, then go to step 3, otherwise call ADDFN (FC, NC, L) and go to step 15.
23. If character FC is a number sign, then a date literal is expected. If $NC \neq 5$, then go to step 3, otherwise call ADDDT (FC, NC, L) to add the date literal to WBF and go to step 15.
24. If character FC is a numeric character, then a numeric literal is expected, and ADDNM (FC, NC, L) is called to add it to WBF. Then go to step 15.

Name:

SEL

Purpose:

To initiate execution of the Rims system; calls UNITS to initialize the array U(14) of /SYSCOM/; calls CIMAIN which initiates interpretation of the command line.

Linkage:

- Calling sequence: Not applicable
- Common blocks used: SYSCOM, EXPCOM, SECCOM
- Subroutines or functions used: CIMAIN, END, UNITS
- Files used: Message file (logical unit 7), command file (logical unit 13)

Input Description:

None

Output Description:

None

Process Description:

1. Use UNITS to initialize the array U(14) of /SYSCOM/.
2. IW=0, initialize command counter, (not used).
3. Initialize file pointers UW and UR to the values stored in U(7) and U(13) of /SYSCOM/.
4. Write 'ENTER COMMAND' on the device designated as the message file.
5. Set IAP=1 of /SYSCOM/, allows printing of the number of sets in the Status Table; If IAP=0, indicates no printing of status table.

~~3-100~~

105

6. Read command line; if UW \neq 7, write the command line on the designated device. If end-of-file is read, go to step 10.
7. Call CIMAIN, to begin interpretation of the command line.
8. IW=IW+1, increment command counter (not used)
9. Go back to step 3.
10. Call END, terminate execution.

Name:

SORTS

Modification Purpose:

To allow the user to order a set of FLOCON records based upon the contents of fields in either the FLOCON or DAPTS records for that set.

Linkage Modification:

- Calling Sequence: CALL SORTS (SET, NF, LIST, SF)
- Common Blocks used: No change
- Subroutines or functions used:
No change
- Files used: No change

Input Description Modification:

SF: integer value, where SF =1 means sort on FLOCON data, SF =2 means sort on either FLOCON or DAPTS data.

Output Description Modification:

No change

Process Description Modification:

For SF = 2 only, the DAPTS record for each appropriate FLOCON record is retrieved. Next the formats for DAPTS or FLOCON records are loaded as needed, a table of sorting names is loaded in proper hierarchical order and a buffer pointer table is also built to point to the proper buffer for data retrieval. Lastly, the data retrieval section is altered to get data from the appropriate buffer by use of the buffer pointer table as an index.

ame:

SORTP

odification Purpose:

To pass an argument to SORTS to indicate the type of sort to perform.

inkage Modification:

- Calling sequence: CALL SORTP(SF)
- Common blocks used: No change
- Subroutines or functions used: No change
- Files used: No change

input Description Modification:

SF = integer variable, indicates which type of sort SORTS is to perform. 1 = normal sort, 2 = joint sort.

input Description Modification:

None

rocess Description Modification:

The argument SF received from JLASYS is passed to SORTS to allow SORTS to perform the appropriate type of sort.

Name: SPCSET

Modification Purpose: To stop the input process when an end-of-file is read as well as a zero record ID.

Linkage Modification:

- Calling sequence: No change
- Common blocks used: No change
- Subroutines or functions used: No change
- Files used: No change

Input Description Modification: No change

Output Description Modification: No change

Process Description Modification: Insert an end-of-file branch to statement number 3 into the statement that reads from the data file.

Name: SQZE

Purpose: To delete extraneous blanks from a character string and build an array of pointers to the commas in the character string.

Linkage:

- Calling sequence: CALL SQZE (INARY, INST, INLEN, OUTARY, OUTST, OUTLEN, COMMAS)
- Common blocks used: None
- Subroutines or functions used: None
- Files used: None

Input Description:

INARY = integer array name; starting location of the array containing the input string
INST = integer variable; character number of INARY at which to begin processing
INLEN = integer variable; number of characters in INARY to be processed
OUTST = integer variable; character number of OUTARY at which to begin storing output
COMMAS = integer array name; contains the Comma Location Table. The first word contains the number of the last used word in the array and should be input containing the value one upon the first call within any one command.

Output Description:

OUTARY = integer array name; starting location of the array to contain the output
OUTLEN = integer variable; number of characters stored in OUTARY
COMMAS = integer array name; contains the Comma Location Table. The first word contains the number of the last used word in the array. The other words contain the character number of OUTARY where commas occur (exclusive of those commas occurring between

pairs of quote marks). The first word will be output as a negative value when an exclamation mark has been encountered and stored as a terminating comma for the command.

Process Description:

1. Counters and pointers are initialized.
2. If the last character of INARY has been passed, go to step 10.
3. If processing is between quote marks (QSET = 2 or 3), check this character for the terminating quote mark, reset QSET to 1 if it is, and go to step 8.
4. If the character is a blank, go to step 9.
5. If the character is a quote mark, set QSET (= 2 for single quote, = 3 for double quote) and go to step 8.
6. If the character is a comma, store the OUTARY pointer in the next available location in COMMAS, increment the pointer to the last used word of COMMAS, and go to step 8.
7. If the character is an exclamation mark, store the OUTARY pointer in the next available location in COMMAS, increment and negate the pointer to the last used word of COMMAS, store a comma in OUTARY, increment the OUTARY pointer, and go to step 10.
8. Transfer the character to OUTARY and increment the OUTARY pointer.
9. Increment the INARY pointer and go to step 2.
10. Store the pointer to the last used word of COMMAS into word one of COMMAS, calculate OUTLEN as the OUTARY pointer minus OUTST, and return to the calling program.

Name: STAEPR

Purpose: To store arithmetic processing data into the normal command table in mathematic hierarchical order.

Linkage:

- Calling sequence: Call STAEPR (VTAB, OPC, FIRST, LAST, TREG)
- Common blocks used: SY3COM
- Subroutines or functions used: None
- Files used: None

Input Description:

VTAB = integer array; contains pointers to variables or literals in the working buffer format table, intermediate storage registers or special integers representing close or open brackets.

OPC = integer array; contains either mathematical operator indicators or special integers representing close or open brackets.

FIRST = integer variable; pointer to first variable in VTAB and OPC to be used for processing.

LAST = integer variable; pointer to last variable in VTAB & OPC to be used in processing.

TREG = integer variable; index pointer into intermediate storage register buffer. used for intermediate data storage.

Output Description: Normal command table filled with appropriate. arithmetic processing data. TREG updated as intermediate storage registers are needed.

Process Description:

A loop is set up to search the entries in the OPC table. Steps 1 thru 3 performed for all entries.

1. The intermediate storage register pointer (TREG) is updated. The OPC entry for the next two adjacent locations is checked for mathematical hierarchy. If they are of equal hierarchy or if the first is of a lesser hierarchy, step 2 is performed, otherwise step 3 is performed.
2. The next normal command table entry is loaded with values from the current and next entry of VTAB, the current value of OPC and the intermediate storage register pointer (TREG). Then the next entry of VTAB is loaded with the intermediate storage register pointer (TREG) and return to step 1.
3. The next normal command table entry is loaded with values from the next and next +1 entry of VTAB, the next +1 value of OPC and the intermediate storage register pointer (TREG). The next +1 entry of VTAB & OPC is loaded with the current value of VTAB & OPC respectively. The next +2 entry of VTAB is loaded with the intermediate storage register pointer (TREG) and return to step 1.

Name: TFORMW

Purpose: To transfer data from a source buffer to the Working Buffer.

Linkage:

- Calling sequence: CALL TFORMW(ROW, MTCTRW)
- Common blocks used: SY2COM, SY3COM
- Subroutines or functions used: SUBSTR
- Files used: None

Input Description: ROW = integer variable; the row number of the source buffer, BUF in /SY2COM/, where the input data is stored.
MTCTRW = integer variable; the row number of the Move Table Control Table, MTCT in /SY3COM/, to be used for control.

Output Description: None

Process Description: The starting row of the Multilevel Move Table, MLMT in /SY3COM/, is retrieved from MTCT (1, MTCTRW). The number of rows of the MLMT to use is retrieved from MTCT (2, MTCTRW) and used to calculate the final row number. Then for each of these rows, (1) a pointer for the Source Buffer Format, SBF in /SY3COM/, is retrieved from the first word of the row of MLMT, (2) a pointer for the Working Buffer Format, WBF in /SY3COM/, is retrieved from the second word of the row of MLMT, and (3) SUBSTR is used to transfer the data from BUF to WBUF, the Working Buffer in /SY3COM/. After the specified number of rows have been processed, a return is made to the calling routine.

Name: TFORMZ

Purpose: To transfer data from the Working Buffer to a target buffer, converting the data representation when needed.

Linkage:

- Calling sequence: CALL TFORMZ(ROW, PF)
- Common blocks used: SY2COM, SY3COM
- Subroutines or functions used: SUBSTR, DTEINT, CHAR, COMSTR
- Files used: None

Input Description: ROW = integer variable; the row number of the target buffer, BUF in /SY2COM/, where the data is to be stored.
PF = integer variable; indicator for which fields of the target buffer are to be filled from the Working Buffer, WBUF in /SY3COM/. A field is filled if word one of its Target Buffer Format, TBF in /SY3COM/, is equal to PF.

Output Description: None

Process Description: For each row of TBF, the following process is done, and then a return is made to the calling routine:

1. If column 1 of TBF is not equal to PF, ignore this row and go to step 25.
2. Retrieve F, the pointer to the WBF row number, from column 2 of TBF.
3. If the output field type, column 5 of TBF, is > 3, go to step 9.
4. If column 2 of row F of WBF indicates text type (by \$T), then use SUBSTR to transfer the text from the command line array, CMD in /SY3COM/, to BUF and then go to step 25.

5. If column 2 of row F of WBF does not indicate a results field (by \$R), then go to step 7.
6. If column 5 of row F of WBF does not indicate text type (contains non-zero), then go to step 8.
7. Use SUBSTR to transfer the data from WBUF to BUF and if WBUF was a results field, then use SUBSTR to reinitialize WBUF from the first word of row F of WBF and go to step 25, otherwise just go to step 25.
8. Use SUBSTR to transfer the data in WBUF to an integer variable named RESULT and based on the target type (column 5 of TBF), use CHAR (type = 1) or DTEINT (type = 2) to convert RESULT to a character string in BUF. Then reinitialize WBUF from the first word of row F of WBF and go to step 25.
9. Extract the first character of the field in WBUF and store this character in CRDTYP.
10. If the target field type \neq 5, go to step 12.
11. Search the array FLMTYP until a match with CRDTYP is found at element L. If no match is found, set L = 8. Store the 12 characters of row L of table FLMTAB into the field in the target buffer and go to step 25.
12. If the target field type \neq 4, go to step 23.
13. Search the array CMPTYP until a match with CRDTYP is found at element L. If no match is found, set L = 11.
14. If L < 5 or = 11, go to step 22.
15. If L > 7, go to step 19.
16. Extract the pointer to the "UNLOAD" field from TBF(4,1) and store it in K.

17. If $K = 0$, go to step 22.
18. If the "UNLOAD" field is non-blank, then increment L by 3 and go to step 22, otherwise go to step 22.
19. Extract the pointer to the "LSD" field from TBF(4,2) and store it in K .
20. If $K = 0$, then set $L = 11$ and go to step 22.
21. Use SUBSTR to transfer the contents of the "LSD" field from WBUF to the target field and go to step 25.
22. Use SUBSTR to store the 12 characters of row L of table CMPTAB into the field in the target buffer and go to step 25.
23. If the target field type $\neq 9$, go to step 7.
24. Search the array GCMTYP until a match with CRDTYP is found at element L . If no match is found, set $L = 1$. Use SUBSTR to store the 12 characters of row L of table GCMTAB into the field in the target buffer.
25. Move to the next row of TBF and start over at step 1.

Name: TJUMP

Purpose: To eliminate headers and other data associated with a null set.

Linkage:

- Calling sequence: CALL TJUMP
- Common blocks used: SYSCOM, SYZCOM
- Subroutines or functions used: INDEX, SUBSTR, INPARM
- Files used: None

Input Description: A command line containing the set to be checked.

Output Description: None

Process Description: The set in question is checked to see if it contains data. If it does contain data no action is required and the routine exits. If there is no data in the set, the label from the command line is saved for later use. Next the input file is read until a label card containing "LA" followed by the label saved from the JT command line is found. The routine then exits.

Name: UNITS

Purpose: To initialize the integer variable array
U of /SYSCOM/

Linkage:

- Calling sequence: Call units
- Common blocks used: SYSCOM
- Subroutines or functions used: ASSIGN, CLOSE, LAST
- Files used: U(7), U(8), U(11),
U(12), U(13), UNITS.SAT

Input Description: None

Output Description: None

Process Description:

1. Call ASSIGN associates logical unit 1 with disk file UNITS.SAT.
2. Read integer values from UNITS.SAT and store data into their respective storage locations U(7), U(8), U(11), U(12) and U(13).
3. Read the file number, and file name. At END go to step 5.
4. Use LAST to determine the number of characters contained in the file name, then return to step 3.
5. Use CLOSE to close logical unit 1.
6. Return to calling routine.

3.2.2 PROGRAM CROSS-REFERENCE

On the following pages is a cross-reference listing which, for each program used in RIMS, shows all the programs that it calls (both directly from itself and indirectly through programs it calls) and all the programs that call it.

CROSS-REFERENCE PRODUCED AT 16:58:41 ON 31-JAN-78

ACCNO DIRECTLY CALLS:
INDEX INPARN
ACCNO DIRECTLY AND INDIRECTLY CALLS:
COMSTR INDEX INPARN VERIFY
ACCNO IS DIRECTLY CALLED BY:
ADDRC SELREC SPCSET
ADDDT DIRECTLY CALLS:
DTINT
ADDDT DIRECTLY AND INDIRECTLY CALLS:
CHAR COMSTR DTINT INDEX INPARN SUBSTR VERIFY
ADDDT IS DIRECTLY CALLED BY:
RCLPR RCLPR
ADDFN DIRECTLY CALLS:
COMSTR SUBSTR
ADDFN DIRECTLY AND INDIRECTLY CALLS:
COMSTR SUBSTR
ADDFN IS DIRECTLY CALLED BY:
AEPR CIRP RCLPR RCLPR
ADDKEY DIRECTLY CALLS:
APSEL GET LOCATE PUT SUBSTR VERIFY
ADDKEY DIRECTLY AND INDIRECTLY CALLS:
APSEL COMSTR FMSFT GET INDEX INPARN LOCATE LOR NHITS PUT RANGE ROLL SETIN1 SETIN2
SETOUT STATUS SUBSTR VERIFY XXIN1 XXIN2 XXOUT
ADDKEY IS DIRECTLY CALLED BY:
SUPOST CISUR6
ADDT DIRECTLY CALLS:
NO ONE
ADDT DIRECTLY AND INDIRECTLY CALLS:
NO ONE
ADDT IS DIRECTLY CALLED BY:
CIRP RCLPR RCLPR
ADDM DIRECTLY CALLS:
INPARN
ADDM DIRECTLY AND INDIRECTLY CALLS:
COMSTR INDEX INPARN VERIFY
ADDM IS DIRECTLY CALLED BY:
RCLPR RCLPR
ADDR DIRECTLY CALLS:
GET LOCREC PUT SUBSTR
ADDR DIRECTLY AND INDIRECTLY CALLS:
GET LOCREC PUT ROLL SUBSTR
ADDR IS DIRECTLY CALLED BY:
ADDRFC AUFLE CMUTE FORM SPLUS
ADDRFC DIRECTLY CALLS:
ACCNO ADDR CLOSEP INDEX INPARN SELREC SUBSTR
ADDRFC DIRECTLY AND INDIRECTLY CALLS:
ACCNO ADDR CLOSEP COMSTR ENDSET GET INDEX INPARN LOCREC NHITS PUT ROLL SELREC SETOUT
STATUS SUBSTR VERIFY XXOUT
ADDRFC IS DIRECTLY CALLED BY:
CISUR5
AEINT DIRECTLY CALLS:
COMSTR INPARN SOZE
AEINT DIRECTLY AND INDIRECTLY CALLS:
COMSTR INDEX INPARN SOZE VERIFY
AEINT IS DIRECTLY CALLED BY:
CFCR JDFCR JNSCR JPRPCR
AEPR DIRECTLY CALLS:
ADDFN COMSTR DTINT INDEX INPARN STAEPR VERIFY
AEPR DIRECTLY AND INDIRECTLY CALLS:

ORIGINAL PAGE IS
OF POOR QUALITY

ADDFN CHAR COMSTR RTEINT INDEX INPARM STAEPB SUBSTR VERIFY
 AEPB IS DIRECTLY CALLED BY:
 CTRP RLCLPR RPCLPR
 AEPB DIRECTLY CALLS:
 NO ONE.
 AEPB DIRECTLY AND INDIRECTLY CALLS:
 NO ONE.
 APSCT IS DIRECTLY CALLED BY:
 AUFIL CFCR RSTBX
 APSCT DIRECTLY CALLS:
 GET INDEX INPARM LOCATE NHITS RANGE SUBSTR VERIFY
 APSCT DIRECTLY AND INDIRECTLY CALLS:
 COMSTR ENDSET GET INDEX INPARM LOCATE LOR NHITS PUT RANGE ROLL SETINI SETIN2 SETOUT
 STATUS SUBSTR VERIFY XXIN1 XXIN2 XXOUT
 APSCT IS DIRECTLY CALLED BY:
 ADDKEY AUPOST
 APSCT DIRECTLY CALLS:
 NO ONE.
 APSCT DIRECTLY AND INDIRECTLY CALLS:
 NO ONE.
 APSCT IS DIRECTLY CALLED BY:
 AUFIL CFCR RSTBX
 APSCT DIRECTLY CALLS:
 PUT SUBSTR
 APSCT DIRECTLY AND INDIRECTLY CALLS:
 PUT ROLL SUBSTR
 APSCT IS DIRECTLY CALLED BY:
 AUFIL CFCR RSTBX
 ATTACH DIRECTLY CALLS:
 SUBSTR
 ATTACH DIRECTLY AND INDIRECTLY CALLS:
 SUBSTR
 ATTACH IS DIRECTLY CALLED BY:
 BEGIN
 AUFIL DIRECTLY CALLS:
 ADDR APSCT APSINT APSUP AUPOST GETREC INPARM LMTAB REPR SUBSTR TFORM
 AUFIL DIRECTLY AND INDIRECTLY CALLS:
 ADDKEY ADDR APSCT APSFL APSINT APSUP AUPOST CHAR COMSTR DELKEY ENDSET GET GETREC INDEX
 INPARM KKEY LAND LMTAB LNOT LOCATE LOCREC LODFMT LOCREC LOR NHITS PART POST PUT
 RANGE REPR ROLL SETINI SETIN2 SETOUT SSORT STATUS SUBSTR TFORM VERIFY XXIN1 XXIN2 XXOUT
 AUFIL IS DIRECTLY CALLED BY:
 CAUFIL
 AUPOST DIRECTLY CALLS:
 ADDKEY APSFL COMSTR DELKEY GET POST PUT SSORT SUBSTR
 AUPOST DIRECTLY AND INDIRECTLY CALLS:
 ADDKEY APSFL COMSTR DELKEY ENDSET GET INDEX INPARM KKEY LAND LNOT LOCATE LOR NHITS
 PART POST PUT RANGE ROLL SETINI SETIN2 SETOUT SSORT STATUS SUBSTR VERIFY XXIN1 XXIN2
 XXOUT
 AUPOST IS DIRECTLY CALLED BY:
 AUFIL CFCR RSTBX
 BEGIN DIRECTLY CALLS:
 ATTACH COMSTR STATUS SUBSTR
 BEGIN DIRECTLY AND INDIRECTLY CALLS:
 ATTACH COMSTR INDEX INPARM STATUS SUBSTR VERIFY
 BEGIN IS DIRECTLY CALLED BY:
 CTRP
 RLCLPR DIRECTLY CALLS:
 COMSTR
 RLCLPR DIRECTLY AND INDIRECTLY CALLS:
 COMSTR
 RLCLPR IS DIRECTLY CALLED BY:
 JFDCR
 CAUFIL DIRECTLY CALLS:
 AUFIL INDEX INPARM LODFMT
 CAUFIL DIRECTLY AND INDIRECTLY CALLS:
 ADDKEY ADDR APSCT APSFL APSINT APSUP AUFIL AUPOST CHAR COMSTR DELKEY ENDSET GET GETREC

120

INDEX INPARM KOMKEY LAND LMTAB LNOT LOCATE LOCREC LODFMT LODREC LOR NHITS PART POST
PUT RANGE REPR ROLL SETIN1 SETIN2 SETOUT SSOBY STATUS SUBSTR TFORM VERIFY XXIN1 XXIN2

XXOUT

CAUFIL IS DIRECTLY CALLED BY:

CISUB6

CFCR DIRECTLY CALLS:

AEINIT	APSCNT	APSINT	APSTUP	AUPOST	CICFDF	EXCMDS	FTCMP	FTFMT	GETREC	REPR	SETIN1	TFORMW	TFORMZ
--------	--------	--------	--------	--------	--------	--------	-------	-------	--------	------	--------	--------	--------

XXIN1

CFCR DIRECTLY AND INDIRECTLY CALLS:

ADDDT	ADDEN	ADKEY	ADLT	ADNM	AEINIT	AEPR	APSCNT	APSEL	APSINT	APSTUP	AUPOST	CFIND	CHAR
CICFDF	COMSTR	DELKEY	DTEINT	ENDSET	EXCMD	EXCMDS	FTCMP	FTFMT	GET	GETREC	INDEX	INPARM	KOMKEY
LAND	LNOT	LOCATE	LOCREC	LODFMT	LODREC	LOR	NHITS	PART	POST	PRINTD	PUT	RANGE	REPR
RLCLPR	ROLL	RPCLPR	SETIN1	SETIN2	SETOUT	SOZE	SSORT	STAEP	STATUS	SUBSTR	TFORMW	TFORMZ	VERIFY

XXIN1 XXIN2 XXOUT

CFCR IS DIRECTLY CALLED BY:

CISUB1

CFIND DIRECTLY CALLS:

INDEX

CFIND DIRECTLY AND INDIRECTLY CALLS:

COMSTR INDEX

CFIND IS DIRECTLY CALLED BY:

CICFDF CIRC PLCLPR RPCLPR

CHAR DIRECTLY CALLS:

NO ONE

CHAR DIRECTLY AND INDIRECTLY CALLS:

NO ONE

CHAR IS DIRECTLY CALLED BY:

DTEINT EXCMD MOVSEG TFORM TFORMZ

CICFDF DIRECTLY CALLS:

CFIND PLCLPR RPCLPR

CICFDF DIRECTLY AND INDIRECTLY CALLS:

ADDDT	ADDEN	ADLT	ADNM	AEPR	CFIND	CHAR	COMSTR	DTEINT	INDEX	INPARM	RLCLPR	RPCLPR	STAEP
-------	-------	------	------	------	-------	------	--------	--------	-------	--------	--------	--------	-------

SUBSTR VERIFY

CICFDF IS DIRECTLY CALLED BY:

CFCR JDFCFC

CIMAIN DIRECTLY CALLS:

CISUB1	CISUB2	CISUB3	CISUB4	CISUB5	CISUB6	CLOSEP	JPRPCR	MODE	STATUS	SUBSTR	VERIFY
--------	--------	--------	--------	--------	--------	--------	--------	------	--------	--------	--------

CIMAIN DIRECTLY AND INDIRECTLY CALLS:

ACCD	ADDDT	ADDEN	ADKEY	ADLT	ADNM	ADDR	ADDR	AEINIT	AEPR	APSCNT	APSEL	APSINT	APSTUP
ATTACH	AUFIL	AUPOST	BEIN	BLDTF	CAUFIL	CFCR	CFIND	CHAR	CICFDF	CIRC	CISUB1	CISUB2	CISUB3
CISUB4	CISUB5	CISUB6	CLEANP	CLOSE	CLOSEP	CMORI	CMUTE	COMBIN	COMSTR	CRESTS	DBPRO	DELETE	DELKEY
DELD	DEIREC	DISMT	DISPDD	DISPLA	DTEINT	END	ENDSET	EXCMD	EXCMDS	EXPAND	FIND	FORM	FORMM
FTCMP	FTFMT	GET	GETCLO	GETPAR	GETREC	HEADER	INDEX	INPARM	JDFCFC	JNSNCR	JPRPCR	KOMKEY	LAND
LAST	LMTAB	LNOT	LOCATE	LOCREC	LODFMT	LODREC	LOR	LXOR	MODE	MOVSEG	NHITS	PARSEC	PAPSEP
PART	POST	PRINTD	PSWRD	PUT	RANGE	REAF	REPKEY	REPR	REPREC	RESTRX	RLCLPR	ROLL	RPCLPR
SELECT	SELREC	SETIN1	SETIN2	SETOUT	SETINIV	SINUS	SORTP	SCRTS	SPCSET	SPLUS	SOZE	SSORT	STAEP
STATUS	STCAT	SURSTR	TFORM	TFORMW	TFORMZ	TJUMP	UNLOCK	VERIFY	XPOST	XREF	XXIN1	XXIN2	XXOUT

CIMAIN IS DIRECTLY CALLED BY:

SEL

CIRC DIRECTLY CALLS:

ADDDT	ADLT	AEPR	CFIND	COMSTR	FTCMP	FTFMT	INDEX	INPARM
-------	------	------	-------	--------	-------	-------	-------	--------

CIRC DIRECTLY AND INDIRECTLY CALLS:

ADDDT	ADLT	AEPR	CFIND	CHAR	COMSTR	DTEINT	FTCMP	FTFMT	GET	INDEX	INPARM	LOCREC	LODFMT
-------	------	------	-------	------	--------	--------	-------	-------	-----	-------	--------	--------	--------

PRINTD ROLL STAEP SUBSTR VERIFY

CIRC IS DIRECTLY CALLED BY:

JPRPCR

CISUB1 DIRECTLY CALLS:

CFCR END JDFCFC

CISUB1 DIRECTLY AND INDIRECTLY CALLS:

ADDDT	ADDEN	ADKEY	ADLT	ADNM	AEINIT	AEPR	APSCNT	APSEL	APSINT	APSTUP	AUPOST	BLDTF	CFCR
CFIND	CHAR	CICFDF	COMSTR	DELKEY	DISMT	DTEINT	END	ENDSET	EXCMD	EXCMDS	FTCMP	FTFMT	GET
GETREC	INDEX	INPARM	JDFCFC	KOMKEY	LAND	LNOT	LOCATE	LOCREC	LODFMT	LOR	NHITS	PART	
POST	PRINTD	PUT	RANGE	REPR	RLCLPR	ROLL	RPCLPR	SETIN1	SETIN2	SETOUT	SOZE	SSORT	STAEP

STATUS SURSTR TFORMW TFORMZ VERIFY XXIN1 XXIN2 XXOUT

CISUB1 IS DIRECTLY CALLED BY:

CIMAIN

CISUPP DIRECTLY CALLS:
 COMSTR DELETE HEADER JNSNCR SELECT STCNT
 CISUPP DIRECTLY AND INDIRECTLY CALLS:
 ADDOT ADDFN ADPLT ADDNM AETNIT AEPR CFIND CHAR COMBIN COMSTR DELETE DTEINT ENDSET EXCMD
 EYCMOS FTCHD FTFMT GET GETREC HEADER INDEX INPARM JNSNCR LAND LNOT LOCATE LOCREC LODFMT
 LODREC LOR LXOR NHITS PRINTD PUT RANGE RLCPR ROLL SELECT SETINI SETIN2 SETOUT SIZE
 STAEPP STATUS STCNT SUBSTR TFORMM VERIFY XXIN1 XXIN2 XXOUT
 CISUP2 IS DIRECTLY CALLED BY:
 CMAIN
 CISUP3 DIRECTLY CALLS:
 REGIN CLEANP SMINUS PLUS UNLOCK
 CISUP3 DIRECTLY AND INDIRECTLY CALLS:
 ADDR ATTACH REGIN CLEANP COMSTR DELR GET INDEX INPARM LOCREC LODREC PSWRD PUT ROLL
 SMINUS PLUS STATUS SUBSTR UNLOCK VERIFY
 CISUB3 IS DIRECTLY CALLED BY:
 CMAIN
 CISUB4 DIRECTLY CALLS:
 CMDRI MOVSEF SORTP TJUMP
 CISUB4 DIRECTLY AND INDIRECTLY CALLS:
 ADDR CHAR CLOSEP CMDRI CMPTU COMSTR GET GETREC INDEX INPARM KOMKEY LAST LMVTAB
 LOCREC LODFMT LODREC MOVSEF PART PUT REPR ROLL SETINI SETOUT SORTP SORTS SSORT SUBSTR
 TJUMP VERIFY XXIN1 XXOUT
 CISUB4 IS DIRECTLY CALLED BY:
 CMAIN
 CISUB5 DIRECTLY CALLS:
 ADDR DISPD DISPLA EXPAND FORMM PARSEC PARSEP REAF SELREC SEUNIV SPCSET XREF
 CISUB5 DIRECTLY AND INDIRECTLY CALLS:
 ACCND ADDR ADDPEC CLOSEP COMSTR DISPD DISPLA ENDSET EXPAND FIND FORM FORMM GET GETCLD
 GETPAR INDEX INPARM LOCREC LODFMT LODREC NHITS PARSEC PARSEP PUT REAF ROLL SELREC SETINI
 SETOUT SEUNIV SPCSET STATUS SUBSTR VERIFY XREF XXIN1 XXOUT
 CISUB5 IS DIRECTLY CALLED BY:
 CMAIN
 CISUB6 DIRECTLY CALLS:
 ADDKEY CAUFIL CRESTS DBPRO DELKEY DELREC REPKEY REPREC XPOST
 CISUB6 DIRECTLY AND INDIRECTLY CALLS:
 ADDKEY ADDR APSCNT APSEL APSINT APSUP AUFILE AUPOST CAUFIL CHAR CLOSEP COMSTR CRESTS DBPRO
 DELKEY DELR DELREC ENDSET GET GETREC INDEX INPARM KOMKEY LAND LMVTAB LNOT LOCATE LOCREC
 LODFMT LODREC LOP NHITS PART POST PUT RANGE REPKEY REPR REPREC RESRXX ROLL SETINI
 SETIN2 SETOUT SSORT STATUS SUBSTR TFORM VERIFY XPOST XXIN1 XXIN2 XXOUT
 CISUB6 IS DIRECTLY CALLED BY:
 CMAIN
 CLEANP DIRECTLY CALLS:
 NO ONE
 CLEANP DIRECTLY AND INDIRECTLY CALLS:
 NO ONE
 CLEANP IS DIRECTLY CALLED BY:
 CISUB3
 CLOSEP DIRECTLY CALLS:
 COMSTR LAST SUBSTR
 CLOSEP DIRECTLY AND INDIRECTLY CALLS:
 COMSTR LAST SUBSTR
 CLOSEP IS DIRECTLY CALLED BY:
 SORTP
 CLOSEP DIRECTLY CALLS:
 NO ONE
 CLOSEP DIRECTLY AND INDIRECTLY CALLS:
 NO ONE
 CLOSEP IS DIRECTLY CALLED BY:
 ADDR CMAIN CMPTU CRESTS FORM REPKEY REPREC RESRXX XPOST
 CMDRI DIRECTLY CALLS:
 CMPTU INDEX INPARM SUBSTR
 CMDRI DIRECTLY AND INDIRECTLY CALLS:
 ADDR CLOSEP CMPTU COMSTR GET GETREC INDEX INPARM LMVTAB LOCREC LODFMT LODREC PUT REPR
 ROLL SETINI SUBSTR VERIFY XXIN1
 CMDRI IS DIRECTLY CALLED BY:
 CISUB4

ORIGINAL PAGE IS
 OF POOR QUALITY

COMPUTE DIRECTLY CALLS:
 ADDR CLOSEP GETREC INPARM LMVTAB LODREC REPR SETIN1 SUBSTR VERIFY XXIN1
 COMPUTE DIRECTLY AND INDIRECTLY CALLS:
 ADDR CLOSEP COMSTR GET GETREC INDEX INPARM LMVTAB LOCREC LODFMT LODREC PUT REPR ROLL
 SETIN1 SUBSTR VERIFY XXIN1
 COMPUTE IS DIRECTLY CALLED BY:
 CWDRI
 COMBIN DIRECTLY CALLS:
 ENDSET INDEX INPARM LAND LNOT LOP LXOR SETIN1 SETIN2 SETOUT SUBSTR VERIFY
 COMBIN DIRECTLY AND INDIRECTLY CALLS:
 COMSTR ENDSET GET INDEX INPARM LAND LNOT LOR LXOR NHITS PUT ROLL SETIN1 SETIN2
 SETOUT STATUS SUBSTR VERIFY XXIN1 XXIN2 XXOUT
 COMBIN IS DIRECTLY CALLED BY:
 CYSUR2
 COMSTR DIRECTLY CALLS:
 NO ONE
 COMSTR DIRECTLY AND INDIRECTLY CALLS:
 NO ONE
 COMSTR IS DIRECTLY CALLED BY:
 ADDFN AFINIT AEPR AUPOST BEGIN BLDTBF CIRP CLOSEL DBPRO EXCHD EXPAND FIND EORM LTFMT
 INDEX JPRPCR KMKKEY LAST LMVTAB LOCATE RANGE RLCLPR RPCLPR TFORM TFORMZ UNLOCK
 CRESTS DIRECTLY CALLS:
 CLOSEP INDEX INPARM LODFMT RESTRX
 CRESTS DIRECTLY AND INDIRECTLY CALLS:
 ADDKEY APSCNT APSEL APSINT APSTUP AUPOST CHAR CLOSEP COMSTR DELKEY DELR ENDSET GET GETREC
 INDEX INPARM KMKKEY LAND LMVTAB LNOT LOCATE LOCREC LODFMT LODREC LOR NHITS PART POST
 PUT RANGE REPR RESTRX ROLL SETIN1 SETIN2 SETOUT SSORT STATUS SUBSTR TFORM VERIFY XXIN1
 XXIN2 XXOUT
 CRESTS IS DIRECTLY CALLED BY:
 CYSUR6
 DBPRO DIRECTLY CALLS:
 COMSTR SUBSTR VERIFY
 DBPRO DIRECTLY AND INDIRECTLY CALLS:
 COMSTR INDEX SUBSTR VERIFY
 DBPRO IS DIRECTLY CALLED BY:
 CYSUR6
 DELETE DIRECTLY CALLS:
 INPARM
 DELETE DIRECTLY AND INDIRECTLY CALLS:
 COMSTR INDEX INPARM VERIFY
 DELETE IS DIRECTLY CALLED BY:
 CYSUR2
 DELKEY DIRECTLY CALLS:
 LOCATE PUT SUBSTR VERIFY
 DELKEY DIRECTLY AND INDIRECTLY CALLS:
 COMSTR GET INDEX LOCATE PUT ROLL SUBSTR VERIFY
 DELKEY IS DIRECTLY CALLED BY:
 AUPOST CYSUR6
 DELR DIRECTLY CALLS:
 LOCREC PUT
 DELR DIRECTLY AND INDIRECTLY CALLS:
 GET LOCREC PUT ROLL
 DELR IS DIRECTLY CALLED BY:
 DELREC RESTRX SMINIS
 DELREC DIRECTLY CALLS:
 DELR INPARM
 DELREC DIRECTLY AND INDIRECTLY CALLS:
 COMSTR DELR GET INDEX INPARM LOCREC PUT ROLL VERIFY
 DELREC IS DIRECTLY CALLED BY:
 CYSUR6
 DISFMT DIRECTLY CALLS:
 NO ONE
 DISFMT DIRECTLY AND INDIRECTLY CALLS:
 NO ONE
 DISFMT IS DIRECTLY CALLED BY:
 JPRPCR

DISPDD DIRECTLY CALLS:
INDEX INPARN LOOFMT
DISPDD DIRECTLY AND INDIRECTLY CALLS:
COMSTR GET INDEX INPARN LOCREC LOOFMT ROLL VERIFY
DISPDD IS DIRECTLY CALLED BY:
CISURS
DISPLA DIRECTLY CALLS:
INDEX INPARN LOOREC SETINI XXINI
DISPLA DIRECTLY AND INDIRECTLY CALLS:
COMSTR GET INDEX INPARN LOCREC LOOREC ROLL SETINI VERIFY XXINI
DISPLA IS DIRECTLY CALLED BY:
CISURS
DTEINT DIRECTLY CALLS:
CHAR INPARN SUBSTR
DTEINT DIRECTLY AND INDIRECTLY CALLS:
CHAR COMSTR INDEX INPARN SUBSTR VERIFY
DTEINT IS DIRECTLY CALLED BY:
ADDCT AEPR EXCMD FORMZ
END DIRECTLY CALLS:
NO ONE
END DIRECTLY AND INDIRECTLY CALLS:
NO ONE
END IS DIRECTLY CALLED BY:
CISURI SEI
ENDSET DIRECTLY CALLS:
INPARN NHITS SURSTR XXOUT
ENDSET DIRECTLY AND INDIRECTLY CALLS:
COMSTR INDEX INPARN NHITS PUT ROLL STATUS SUBSTR VERIFY XXOUT
ENDSET IS DIRECTLY CALLED BY:
COMBIN GETCLO GETPAR INSNCR MODE RANGE SELECT SEIREC SPCSET
EXCMD DIRECTLY CALLS:
CHAR COMSTR DTEINT INPARN SUBSTR
EXCMD DIRECTLY AND INDIRECTLY CALLS:
CHAR COMSTR DTEINT INDEX INPARN SUBSTR VERIFY
EXCMD IS DIRECTLY CALLED BY:
EXCMD5
EXCMD5 DIRECTLY CALLS:
EXCMD SUBSTR
EXCMD5 DIRECTLY AND INDIRECTLY CALLS:
CHAR COMSTR DTEINT EXCMD INDEX INPARN SUBSTR VERIFY
EXCMD5 IS DIRECTLY CALLED BY:
CECR JEDEC INSNCR IPPPCR
EXPAND DIRECTLY CALLS:
COMSTR FIND GET SUBSTR
EXPAND DIRECTLY AND INDIRECTLY CALLS:
COMSTR FIND GET ROLL SUBSTR
EXPAND IS DIRECTLY CALLED BY:
CISURS
FIND DIRECTLY CALLS:
COMSTR GET
FIND DIRECTLY AND INDIRECTLY CALLS:
COMSTR GET ROLL
FIND IS DIRECTLY CALLED BY:
EXPAND
FORM DIRECTLY CALLS:
ADDR CLOSE COMSTR
FORM DIRECTLY AND INDIRECTLY CALLS:
ADDR CLOSE COMSTR GET LOCREC PUT ROLL SUBSTR
FORM IS DIRECTLY CALLED BY:
FORMH
FORMH DIRECTLY CALLS:
FORM INDEX INPARN
FORMH DIRECTLY AND INDIRECTLY CALLS:
ADDR CLOSE COMSTR FORM GET INDEX INPARN LOCREC PUT ROLL SUBSTR VERIFY
FORMH IS DIRECTLY CALLED BY:
CISURS

FTCMP DIRECTLY CALLS:
SUBSTR

FTCMP DIRECTLY AND INDIRECTLY CALLS:
SUBSTR

FTCMP IS DIRECTLY CALLED BY:
CFER CIRR JDFCR INSNCR

FTFMT DIRECTLY CALLS:
COMSTR GET INPARM LOCREC LODFMT PRINTID

FTFMT DIRECTLY AND INDIRECTLY CALLS:
COMSTR GET INDEX INPARM LOCREC LODFMT PRINTID ROLL VERIFY

FTFMT IS DIRECTLY CALLED BY:
CFER CIRR JDFCR INSNCR

GET DIRECTLY CALLS:
ROLL

GET DIRECTLY AND INDIRECTLY CALLS:
ROLL

GET IS DIRECTLY CALLED BY:
ADKEY ADDR APSEL APOST EXPAND FIND FTFMT GETCLO LOCATE LOCREC LODFMT LODRE PART POST
RANGE REPKFY REPR SELECT SEUNIV SORTS SSORT XXINI XXIN2

GETCLO DIRECTLY CALLS:
ENDSET GET LOCREC SETINI SETOUT XXINI XXOUT

GETCLO DIRECTLY AND INDIRECTLY CALLS:
COMSTR ENDSET GET INDEX INPARM LOCREC NHITS PUT ROLL SETINI SETOUT STATUS SUBSTR VERIFY
XXINI XXOUT

GETCLO IS DIRECTLY CALLED BY:
PARSEC

GETPAR DIRECTLY CALLS:
ENDSET SETINI SETOUT XXINI XXOUT

GETPAR DIRECTLY AND INDIRECTLY CALLS:
COMSTR ENDSET GET INDEX INPARM NHITS PUT ROLL SETINI SETOUT STATUS SUBSTR VERIFY XXINI
XXOUT

GETPAR IS DIRECTLY CALLED BY:
PARSEP

GETREC DIRECTLY CALLS:
INPARM LODFMT LOCREC

GETREC DIRECTLY AND INDIRECTLY CALLS:
COMSTR GET INDEX INPARM LOCREC LODFMT LODREC ROLL VERIFY

GETREC IS DIRECTLY CALLED BY:
AUFIL CFER CMPTU JDFCR INSNCR JPRPCR MOVSEG RESTPX SORTS

HEADER DIRECTLY CALLS:
INDEX INPARM SUBSTR

HEADER DIRECTLY AND INDIRECTLY CALLS:
COMSTR INDEX INPARM SUBSTR VERIFY

HEADER IS DIRECTLY CALLED BY:
CISUR2

INDEX DIRECTLY CALLS:
COMSTR

INDEX DIRECTLY AND INDIRECTLY CALLS:
COMSTR

INDEX IS DIRECTLY CALLED BY:
ACCD ADDR EC APR APSEL CAUFIL CFIND CIRR CMDRI COMBIN CRESTS DISPOD DISPLA FORMH HEADER
INPARM MOVSEG RANGE REAF REPKFY REPREC RLCLPR RLCLPR SELECT SORTP SPCSET SICNT TJUMP VERIFY
XPOST

INPARM DIRECTLY CALLS:
INDEX VERIFY

INPARM DIRECTLY AND INDIRECTLY CALLS:
COMSTR INDEX VERIFY

INPARM IS DIRECTLY CALLED BY:
ACCD ADDR ADDR EC APR APSEL AUFIL CAUFIL CIRR CMDRI CMPTU COMBIN CRESTS DELETE
DELDFC DISPOD DISPLA DTENT ENDSET EXCHD FORMH FTFMT GETREC HEADER MOVSEG PARSEC PARSEP REAF
REPREC SELECT SORTP STATUS STCNT TFORM TJUMP XPOST XREF

JDFCR DIRECTLY CALLS:
AEINIT BLDTRF CICDF DISFMT EXCHDS FTFMT FTFMT GETREC LODFMT PRINTID SETINI TFORM TFORM2 XXINI

JDFCR DIRECTLY AND INDIRECTLY CALLS:
ADDD ADDN ADDN ADDN AEINIT APR BLDTRF CFIND CHAR CICDF COMSTR DISFMT DTENT EXCHD
EXCHDS FTFMT FTFMT GET REC GETREC INDEX INPARM LOCREC LODFMT LODREC PRINTID RLCLPR RLCLPR XPCLP

125

SETINI SOZE STAEPB SUBSTR TFORMW TFORMZ VERIFY. XXINI

JFDECR IS DIRECTLY CALLED BY:

CISUR1

JNSNCR DIRECTLY CALLS:

AFINT ENDSET EXCMD FICMP FIFMT GETREC PRNTID RLCLPR SETINI SETOUT TFORMW XXINI XXOUT

JNSNCR DIRECTLY AND INDIRECTLY CALLS:

ADDN ADDFN ADOLY ADDNM AEINIT AEPR CFIND CHAR COMSTR DTEINT ENDSET EXCMD EXCMD5 FICMP
FIFMT GET GETREC INDEX INPARN LOCREC LODFMT LODREC NHLS PRNTID PUT RLCLPR ROLL SETINI
SETOUT SOZE STAEPB STATUS SUBSTR TFORMW VERIFY XXINI XXOUT

JNSNCR IS DIRECTLY CALLED BY:

CISUR2

JPRPCR DIRECTLY CALLS:

AEINIT CIRP COMSTR EXCMD GETREC PRNTID SETINI SUBSTR TFORMW TFORMZ XXINI

JPRPCR DIRECTLY AND INDIRECTLY CALLS:

ADDN ADOLY AEINIT AEPR CFIND CHAR CIRP COMSTR DTEINT EXCMD EXCMD5 FICMP FIFMT GET
GETREC INDEX INPARN LOCREC LODFMT LODREC PRNTID ROLL SETINI SOZE STAEPB SUBSTR TFORMW TFORMZ
VERIFY XXINI

JPRPCR IS DIRECTLY CALLED BY:

CIMAIN

KOMKEY DIRECTLY CALLS:

COMSTR

KOMKEY DIRECTLY AND INDIRECTLY CALLS:

COMSTR

KOMKEY IS DIRECTLY CALLED BY:

PART

LAND DIRECTLY CALLS:

XXINI XXIN2 XXOUT

LAND DIRECTLY AND INDIRECTLY CALLS:

GET PUT ROLL XXINI XXIN2 XXOUT

LAND IS DIRECTLY CALLED BY:

COMBIN POST

LAST DIRECTLY CALLS:

COMSTR

LAST DIRECTLY AND INDIRECTLY CALLS:

COMSTR

LAST IS DIRECTLY CALLED BY:

CLOSEL UNITS

LMVTAR DIRECTLY CALLS:

COMSTR

LMVTAR DIRECTLY AND INDIRECTLY CALLS:

COMSTR

LMVTAR IS DIRECTLY CALLED BY:

AUFIL CHPUTE RESTRX

LNOT DIRECTLY CALLS:

XXINI XXIN2 XXOUT

LNOT DIRECTLY AND INDIRECTLY CALLS:

GET PUT ROLL XXINI XXIN2 XXOUT

LNOT IS DIRECTLY CALLED BY:

COMBIN POST

LOCATE DIRECTLY CALLS:

COMSTR GET

LOCATE DIRECTLY AND INDIRECTLY CALLS:

COMSTR GET ROLL

LOCATE IS DIRECTLY CALLED BY:

ADDKEY APSEL DELKEY POST RANGE REPKEY SELECT

LOCRC DIRECTLY CALLS:

GET

LOCRC DIRECTLY AND INDIRECTLY CALLS:

GET ROLL

LOCRC IS DIRECTLY CALLED BY:

ADDR DELR FIFMT GETCLD LODFMT LODREC REPR SELREC

LODFMT DIRECTLY CALLS:

GET LOCREC

LODFMT DIRECTLY AND INDIRECTLY CALLS:

GET LOCRFC ROLL

ORIGINAL PAGE IS
OF POOR QUALITY

1210

CAUFIL CRESTS DISPOD FTENT GETREC JFDFCR

LODREC DIRECTLY CALLS:
GET LOCREC

LODREC DIRECTLY AND INDIRECTLY CALLS:
GET LOCREC POLL

LODREC IS DIRECTLY CALLED BY:
COMPUTE DISPLA GETREC INLOCK

LOD DIRECTLY CALLS:
XXIN1 XXIN2 XXOUT

LOD DIRECTLY AND INDIRECTLY CALLS:
GET PUT POLL XXIN1 XXIN2 XXOUT

LOD IS DIRECTLY CALLED BY:
COMBIN LXCR POST RANGE

LXOP DIRECTLY CALLS:
LOR

LXOR DIRECTLY AND INDIRECTLY CALLS:
GET LOR PUT ROLL XXIN1 XXIN2 XXOUT

LXOR IS DIRECTLY CALLED BY:
CCHSTN

MODE DIRECTLY CALLS:
ENDSET

MODE DIRECTLY AND INDIRECTLY CALLS:
COMSTR ENDSET INDEX INPARM NHITS PUT ROLL STATUS SUBSTR VERIFY XXOUT

MODE IS DIRECTLY CALLED BY:
CIVATN

MOVSEG DIRECTLY CALLS:
CHAR GETREC INDEX INPARM REPR

MOVSEG DIRECTLY AND INDIRECTLY CALLS:
CHAR COMSTR GET GETREC INDEX INPARM LOCREC LODFNT LODREC PUT REPR ROLL SUBSTR VERIFY

MOVSEG IS DIRECTLY CALLED BY:
CISUB4

NHITS DIRECTLY CALLS:
STATUS

NHITS DIRECTLY AND INDIRECTLY CALLS:
COMSTR INDEX INPARM STATUS VERIFY

NHITS IS DIRECTLY CALLED BY:
APSEL ENDSET SELECT SEUNIV

PARSEC DIRECTLY CALLS:
GETCLO INPARM

PARSEC DIRECTLY AND INDIRECTLY CALLS:
COMSTR ENDSET GET GETCLO INDEX INPARM LOCREC NHITS PUT ROLL SETIN1 SETOUT STATUS SUBSTR

PARSEC IS DIRECTLY CALLED BY:
CISUB5

PARSEC DIRECTLY CALLS:
GETPAR INPARM

PARSEC DIRECTLY AND INDIRECTLY CALLS:
COMSTR ENDSET GET GETPAR INDEX INPARM NHITS PUT ROLL SETIN1 SETOUT STATUS SUBSTR VERIFY

PARSEC IS DIRECTLY CALLED BY:
CISUB5

PART DIRECTLY CALLS:
GET KOMKEY PUT SUBSTR

PART DIRECTLY AND INDIRECTLY CALLS:
COMSTR GET KOMKEY PUT ROLL SUBSTR

PART IS DIRECTLY CALLED BY:
SSORT

POST DIRECTLY CALLS:
GET LAND INOT LOCATE LOR PUT SETIN1 SETIN2 SETOUT SUBSTR

POST DIRECTLY AND INDIRECTLY CALLS:
COMSTR GET LAND INOT LOCATE LOR PUT ROLL SETIN1 SETIN2 SETOUT SUBSTR XXIN1 XXIN2

POST IS DIRECTLY CALLED BY:
AUPOST YPOST

PRINTC DIRECTLY CALLS:
NO ONE

127

PRINTS DIRECTLY AND INDIRECTLY CALLS:

NO ONE

PRINT IS DIRECTLY CALLED BY:

PFMT JEDEC JNSNCR JPRPCR

PSWD DIRECTLY CALLS:

NO ONE

PSWD DIRECTLY AND INDIRECTLY CALLS:

NO ONE

PSWD IS DIRECTLY CALLED BY:

SYNUS SPLUS UNLOCK

PUT DIRECTLY CALLS:

ROLL

PUT DIRECTLY AND INDIRECTLY CALLS:

ROLL

PUT IS DIRECTLY CALLED BY:

ADDKEY ADDR APSTUP AUPOST DELKEY DELR PART POST REPKEY REPR SEUNIV SORTS SPLUS SSORT

XXOUT

RANGE DIRECTLY CALLS:

COMSTR ENDSET GET INDEX LOCATE LOR SETIN1 SETIN2 SETOUT SUBSTR VERIFY XXIN1 XXOUT

RANGE DIRECTLY AND INDIRECTLY CALLS:

COMSTR ENDSET GET INDEX INPARM LOCATE LOR NHITS PUT ROLL SETIN1 SETIN2 SETOUT STATUS

SUBSTR VERIFY XXIN1 XXIN2 XXOUT

RANGE IS DIRECTLY CALLED BY:

APSEL SELECT

REAF DIRECTLY CALLS:

INDEX INPARM SUBSTR

REAF DIRECTLY AND INDIRECTLY CALLS:

COMSTR INDEX INPARM SUBSTR VERIFY

REAF IS DIRECTLY CALLED BY:

CISUP5

REPKEY DIRECTLY CALLS:

CLOSEP GET INDEX LOCATE PUT SUBSTR VERIFY

REPKEY DIRECTLY AND INDIRECTLY CALLS:

CLOSEP COMSTR GET INDEX LOCATE PUT ROLL SUBSTR VERIFY

REPKEY IS DIRECTLY CALLED BY:

CISUR6

REPR DIRECTLY CALLS:

GET LOCREC PUT SUBSTR

REPR DIRECTLY AND INDIRECTLY CALLS:

GET LOCREC PUT ROLL SUBSTR

REPR IS DIRECTLY CALLED BY:

AMFILE CPCR CHPUTE MOVSEG REPREC RESTRX

REPREC DIRECTLY CALLS:

CLOSEP INDEX INPARM REPR

REPREC DIRECTLY AND INDIRECTLY CALLS:

CLOSEP COMSTR GET INDEX INPARM LOCREC PUT REPR ROLL SUBSTR VERIFY

REPREC IS DIRECTLY CALLED BY:

CISUR6

RESTRX DIRECTLY CALLS:

APSCAT APSINT APSTUP AUPOST CLOSEP DELR GEIREC LMVTAB REPR SETIN1 SUBSTR TFORM XXIN1

RESTRX DIRECTLY AND INDIRECTLY CALLS:

ADDKEY APSCAT APSEL APSINT APSTUP AUPOST CHAR CLOSEP COMSTR DELKEY DELR ENDSET GET GETREC

INDEX INPARM KOKKFY LAND LMVTAB LNDFM LOCATE LOCLOC LOR NHITS PART POST

PUT RANGE REPR ROLL SETIN1 SETIN2 SETOUT SSORT STATUS SUBSTR TFORM VERIFY XXIN1 XXIN2

XXOUT

RESTRX IS DIRECTLY CALLED BY:

CRESTS

RLCLPR DIRECTLY CALLS:

ADDOT ADDFN ADDLT ADDNM AEPR CFIND COMSTR INDEX

RLCLPR DIRECTLY AND INDIRECTLY CALLS:

ADDOT ADDFN ADDLT ADDNM AEPR CFIND CHAR COMSTR DTEINT INDEX INPARM STAEPRE SUBSTR VERIFY

RLCLPR IS DIRECTLY CALLED BY:

CICFDF JNSNCR

ROLL DIRECTLY CALLS:

NO ONE

ROLL DIRECTLY AND INDIRECTLY CALLS:

ORIGINAL PAGE IS
OF POOR QUALITY

128

NO ONE.
 ROLL IS DIRECTLY CALLED BY:
 GET PUT
 PPCLPP DIRECTLY CALLS:
 ADDT ADDF ADDL ADDNM AEPR CFIND COMSTR INDEX
 RPCLPR DIRECTLY AND INDIRECTLY CALLS:
 ADDT ADDF ADDL ADDNM AEPR CFIND CHAR COMSTR DTEINT INDEX INPARM STALPR SUBSTR VERIFY
 RPCLPR IS DIRECTLY CALLED BY:
 CFCFDF
 SEL DIRECTLY CALLS:
 CIMATN END UNITS
 SEL DIRECTLY AND INDIRECTLY CALLS:
 ACCNO ADDT ADDF ADDKEY ADDL ADDNM ADDR ADDREC AEINIT AEPR APSCNT APSEL APSINT APSTUP
 ATTACH AUFIL AUPOST REGIN BLDTRF CAUFIL CFCR CFIND CHAR CFCFDF CIMAIN CIRP CISUB1 CISUB2
 CISUR3 CISUR4 CISUR5 CISUR6 PLFANP CLOSEP CMORI CMPTU COMBIN COMSTR CRESTS DBPRO DELETE
 DELKEY DELR DELPEC DTSFMT DISPOD DISPLA DTEINT END ENDSET EXCHD EXCHDS EXPAND FIND FORM
 FCRMP FICMP FICMT GET GETCLO GETPAR GETREC HEADER INDEX INPARM JFDFCR JNSACR JPRPCR KOMKEY
 LAND LAST LMVTR LNOT LOCATE LOCREC LODFMT LODREC LOR LXOR MODE MOVSEG NHITS PARSEC
 PARSEP PART POST PRYID PSWRD PUT RANGE REAF REPKEY REPR REPREC RESTRX RLCLPR ROLL
 PPCLPR SELECT SELREC SETIN1 SETIN2 SETOUT SEUNIV SMINUS SORTP SORTS SPCSET SPLUS SUZE SSORT
 STAEPR STATUS STCNT SUBSTR TFORM TFORMW TFORMZ TJUMP UNITS UNLOCK VERIFY XPOST XHEF XXIN1
 XXIN2 XXOUT
 SEL IS DIRECTLY CALLED BY:
 NO ONE
 SELECT DIRECTLY CALLS:
 ENDSET GET INDEX INPARM LOCATE NHITS RANGE SUBSTR VERIFY
 SELECT DIRECTLY AND INDIRECTLY CALLS:
 COMSTR ENDSET GET INDEX INPARM LOCATE LOR NHITS PUT RANGE ROLL SETIN1 SETIN2 SETOUT
 STATUS SUBSTR VERIFY XXIN1 XXIN2 XXOUT
 SELECT IS DIRECTLY CALLED BY:
 CISUR2
 SELREC DIRECTLY CALLS:
 ACCNO ENDSET LOCREC SETOUT XXOUT
 SELREC DIRECTLY AND INDIRECTLY CALLS:
 ACCNO COMSTR ENDSET GET INDEX INPARM LOCREC NHITS PUT ROLL SETOUT STATUS SUBSTR VERIFY
 XXOUT
 SELREC IS DIRECTLY CALLED BY:
 ADDR CISCUR
 SETIN1 DIRECTLY CALLS:
 NO ONE
 SETIN1 DIRECTLY AND INDIRECTLY CALLS:
 NO ONE
 SETIN1 IS DIRECTLY CALLED BY:
 CFCR CMPTU COMBIN DISPLA GETCLO GETPAR JFDFCR JNSACR JPRPCR POST RANGE RESTRX SORTS XREF
 SETIN2 DIRECTLY CALLS:
 NO ONE
 SETIN2 DIRECTLY AND INDIRECTLY CALLS:
 NO ONE
 SETIN2 IS DIRECTLY CALLED BY:
 COMBIN POST RANGE
 SETOUT DIRECTLY CALLS:
 NO ONE
 SETOUT DIRECTLY AND INDIRECTLY CALLS:
 NO ONE
 SETOUT IS DIRECTLY CALLED BY:
 COMBIN GETCLO GETPAR JNSACR POST RANGE SELREC SORTS SPCSET
 SEUNIV DIRECTLY CALLS:
 GET NHITS PUT SUBSTR
 SEUNIV DIRECTLY AND INDIRECTLY CALLS:
 COMSTR GET INDEX INPARM NHITS PUT ROLL STATUS SUBSTR VERIFY
 SEUNIV IS DIRECTLY CALLED BY:
 CISUR5
 SMINUS DIRECTLY CALLS:
 DELR PSWRD
 SMINUS DIRECTLY AND INDIRECTLY CALLS:
 DELR GET LOCREC DBPRO PUT

129

SHINUS IS DIRECTLY CALLED BY:

CISU93

SORTP DIRECTLY CALLS:

CLOSEL INDEX INPARM SORTS SUBSTR

SORTP DIRECTLY AND INDIRECTLY CALLS:

CLOSEL COMSTR GET GETREC INDEX INPARM KOMKEY LAST LOCREC LODFMT LODREC PART PUT ROLL

SETINI SETOUT SORTS SSORT SUBSTR VERIFY XXINI XXOUT

SORTP IS DIRECTLY CALLED BY:

CISUR4

SORTS DIRECTLY CALLS:

GET GETRFC PUT SETINI SETOUT SSORT SUBSTR XXINI XXOUT

SORTS DIRECTLY AND INDIRECTLY CALLS:

COMSTR GET GETRFC INDEX INPARM KOMKEY LOCREC LODFMT LODREC PART PUT ROLL SETINI SETOUT

SSORT SUBSTR VERIFY XXINI XXOUT

SORTS IS DIRECTLY CALLED BY:

SORTP

SPCSET DIRECTLY CALLS:

ACCNO ENDSET INDEX SETOUT SUBSTR XXOUT

SPCSET DIRECTLY AND INDIRECTLY CALLS:

ACCNO COMSTR ENDSET INDEX INPARM NHITS PUT ROLL SETOUT STATUS SUBSTR VERIFY XXOUT

SPCSET IS DIRECTLY CALLED BY:

CISUB5

SPLUS DIRECTLY CALLS:

ADDP PSWRD PUT

SPLUS DIRECTLY AND INDIRECTLY CALLS:

ADDP GET LOCREC PSWRD PUT ROLL SUBSTR

SPLUS IS DIRECTLY CALLED BY:

CISUR3

SOZE DIRECTLY CALLS:

NO ONE

SOZE DIRECTLY AND INDIRECTLY CALLS:

NO ONE

SOZE IS DIRECTLY CALLED BY:

ACINIT

SSORT DIRECTLY CALLS:

GET PART PUT

SSORT DIRECTLY AND INDIRECTLY CALLS:

COMSTR GET KOMKEY PART PUT ROLL SUBSTR

SSORT IS DIRECTLY CALLED BY:

AUPOST SORTS

STAEPR DIRECTLY CALLS:

NO ONE

STAEPR DIRECTLY AND INDIRECTLY CALLS:

NO ONE

STAEPR IS DIRECTLY CALLED BY:

AEPR

STATUS DIRECTLY CALLS:

INPARM

STATUS DIRECTLY AND INDIRECTLY CALLS:

COMSTR INDEX INPARM VERIFY

STATUS IS DIRECTLY CALLED BY:

BEGIN CIMAIN NHITS

STCNT DIRECTLY CALLS:

INDEX INPARM SUBSTR

STCNT DIRECTLY AND INDIRECTLY CALLS:

COMSTR INDEX INPARM SUBSTR VERIFY

STCNT IS DIRECTLY CALLED BY:

CISUR2

SUBSTR DIRECTLY CALLS:

NO ONE

SUBSTR DIRECTLY AND INDIRECTLY CALLS:

NO ONE

SUBSTR IS DIRECTLY CALLED BY:

ACDPN ADDKEY ADDR ADDRFC APSEL APSTIP ATTACH AUFILE AUPOST BEGIN CIMAIN CLOSEL CMDPT CMPUTE
COMSTR DBPRD DELKEY ATEJAT FNDSET EXCHD EXCMOS EXPAND FTCHD HEADER JPPPCR PART POST RANGE
REFR DELKEY REPR DESTRY SELECT SENNA SCNTP SORTS SPCSET STCNT TFCR TFORM TFS-22 TJUMP

ORIGINAL PAGE IS
OF POOR QUALITY

UNLOCK
TFORM DIRECTLY CALLS:
CHAR COMSTR INPARM SUBSTR
TFORM DIRECTLY AND INDIRECTLY CALLS:
CHAR COMSTR INDEX INPARM SUBSTR VERIFY
TFORM IS DIRECTLY CALLED BY:
ACFILE PESTRX
TFORM DIRECTLY CALLS:
SUBSTR
TFORM DIRECTLY AND INDIRECTLY CALLS:
SUBSTR
TFORM IS DIRECTLY CALLED BY:
CFGR JDFGR JNSGR JPRGR
TFORM DIRECTLY CALLS:
CHAR COMSTR DIFINT SUBSTR
TFORM DIRECTLY AND INDIRECTLY CALLS:
CHAR COMSTR DIFINT INDEX INPARM SUBSTR VERIFY
TFORM IS DIRECTLY CALLED BY:
CFGR JDFGR JPRGR
TJUMP DIRECTLY CALLS:
INDEX INPARM SUBSTR
TJUMP DIRECTLY AND INDIRECTLY CALLS:
COMSTR INDEX INPARM SUBSTR VERIFY
TJUMP IS DIRECTLY CALLED BY:
CISUR4
UNITS DIRECTLY CALLS:
LAST
UNITS DIRECTLY AND INDIRECTLY CALLS:
COMSTR LAST
UNITS IS DIRECTLY CALLED BY:
SEL
UNLOCK DIRECTLY CALLS:
COMSTR LODREC PSWRD SUBSTR
UNLOCK DIRECTLY AND INDIRECTLY CALLS:
COMSTR GET LODREC LODREC PSWRD ROLL SUBSTR
UNLOCK IS DIRECTLY CALLED BY:
CISUR3
VERIFY DIRECTLY CALLS:
INDEX
VERIFY DIRECTLY AND INDIRECTLY CALLS:
COMSTR INDEX
VERIFY IS DIRECTLY CALLED BY:
ADDKEY AFPR APSEL CMAIN CMPTU COMBIN DSPRO DELKEY INPARM RANGE REPKEY SELECT
XPOST DIRECTLY CALLS:
CLOSEP INDEX INPARM POST
XPOST DIRECTLY AND INDIRECTLY CALLS:
CLOSEP COMSTR GET INDEX INPARM LAND LNOT LOCATE LOR POST PUT ROLL SETIN1 SETIN2
SETOUT SUBSTR VERIFY XXIN1 XXIN2 XXOUT
XPOST IS DIRECTLY CALLED BY:
CISUR4
XREF DIRECTLY CALLS:
INPARM SETIN1 XXIN1
XREF DIRECTLY AND INDIRECTLY CALLS:
COMSTR GET INDEX INPARM ROLL SETIN1 VERIFY XXIN1
XREF IS DIRECTLY CALLED BY:
CISUR5
XXIN1 DIRECTLY CALLS:
GET
XXIN1 DIRECTLY AND INDIRECTLY CALLS:
GET ROLL
XXIN1 IS DIRECTLY CALLED BY:
CFGR CMPTU DISPIA GETOLD GETPAR JDFGR JNSGR JPRGR LAND LNOT LOR RANGI RESTRX SORTS
XREF
XXIN2 DIRECTLY CALLS:
GET
XXIN2 DIRECTLY AND INDIRECTLY CALLS:

GET ROLL

XXIN2 IS DIRECTLY CALLED BY:

LAND LNOT LOR

XXOUT DIPECTLY CALLS:

PUT

XXOUT DIPECTLY AND INDIRECTLY CALLS:

PUT ROLL

XXOUT IS DIPECTLY CALLED BY:

ENDSET GETCID GETPAR JNSNCR LAND LNOT LOR RANGE SELREC SORTS SPCSET

ORIGINAL PAGE IS
OF POOR
QUALITY

5-229-

132

3.2.3 NEW BUFFER FORMATS, COMMON BLOCKS, AND TABLES

On the following pages are described new buffer formats and tables, along with the new common blocks which contain them.

COMMON BLOCK CLTBL

This common block contains the 50 word array named COMMAS,
which is the Comma Location Table.

COMMA LOCATION TABLE

Table that points to the commas which surround relational, replacement, and BY clauses, and report expressions in the packed input string for the command. It is a one-dimensional array of four-byte integers.

Word 1 - binary integer whose absolute value is the last used word of the Comma Location Table. A negative value is used for the JP & RP commands to indicate that the command terminating exclamation point was encountered by subroutine SQZE.

Words 2-n - binary integers whose values are the character numbers in array CMD of /SY3COM/ where significant syntactical commas occur.

COMMON BLOCK SY3COM

<u>Variable or Array</u>	<u>Usage</u>
CTBL(5,50)	Normal Command Table
FCTBL(5,10)	Function Command Table
WBUF(50)	Working Buffer
WBF(7,50)	Working Buffer Format
SBF(7,20)	Source Buffer Format
TBF(7,30)	Target Buffer Format
MLMT(2,20)	Multilevel Move Table
MTCT(3,2)	Move Table Control Table
BPT(10,5)	BY Processing Table
REG(16)	Intermediate storage registers
CMD(100)	Packed input command
NCTBL	last used row number in CTBL
NFCTBL	last used row number in FCTBL
NWBF	last used row number in WBF
NSBF	last used row number in SBF
NTBF	last used row number in TBF
NMLMT	last used row number in MLMT
NMTCT	last used row number in MTCT
NBPT	last used row number in BPT

COMMAND TABLE

General layout for the normal command table and the function command table. Each row of this five-column table represents an operation to be performed by the execute command subroutine, EXCMD.

Column 1 - 1 word - binary integer pointing to the first operand. A positive number is the row number of the Working Buffer Format. A negative number means an intermediate storage register, and its absolute value tells which register.

Column 2 - 1 word - positive binary integer representing the operation to be performed. See Command Operations Table.

Column 3 - 1 word - binary integer pointing to the second operand for binary operations. Same type pointer as column 1.

Column 4 - 1 word - binary integer pointing to the location where the result of the operation is to be stored. Same type pointer as column 1.

Column 5 - 1 word - binary integer whose value is the row number of this command table to which a jump is made when the current operation cannot be performed due to absence of data in an operand.

COMMAND OPERATIONS TABLE

This table does not exist as an identifiable entity in the software. It is an explanation of what is meant by a row of a command table and a definition of what operation is performed for each operator by the subroutine EXCMD. In the description below, columns 1-4 of a command table are referenced by the terms OPND(1), OPERATOR, OPND(2), and RESULT, respectively, and CFLAG is a logical argument in the call to EXCMD.

<u>OPERATOR</u>	<u>OPERATION PERFORMED</u>
1	ADDITION: OPND(1) + OPND(2)→RESULT
2	SUBTRACTION: OPND(1) - OPND(2)→RESULT
3	MULTIPLICATION: OPND(1) * OPND(2)→RESULT
4	DIVISION: OPND(1)/OPND(2)→RESULT
5	.LT.: IF OPND(1) < OPND(2), THEN .TRUE.→CFLAG, OTHERWISE .FALSE.→CFLAG
6	.LE.: IF OPND(1) ≤ OPND(2), THEN .TRUE.→CFLAG, OTHERWISE .FALSE.→CFLAG
7	.EQ.: IF OPND(1) = OPND(2), THEN .TRUE.→CFLAG, OTHERWISE .FALSE.→CFLAG
8	.NE.: IF OPND(1) ≠ OPND(2), THEN .TRUE.→CFLAG, OTHERWISE .FALSE.→CFLAG
9	.GE.: IF OPND(1) ≥ OPND(2), THEN .TRUE.→ CFLAG, OTHERWISE .FALSE.→CFLAG
10	.GT.: IF OPND(1) > OPND(2), THEN .TRUE.→ CFLAG, OTHERWISE .FALSE.→CFLAG
11	COUNT: IF OPND(2) IS NOT BLANK, THEN OPND(1) +1→ OPND(1)
12	NUM. MIN.: THE NUMERICAL MINIMUM OF OPND(1) AND OPND(2)→ OPND(1)
13	NUM. MAX.: THE NUMERICAL MAXIMUM OF OPND(1) AND OPND(2)→ OPND(1)

<u>OPERATOR</u>	<u>OPERATION PERFORMED</u>
14	ALPHA MIN.: THE ALPHA MINIMUM OF OPND(1) AND OPND(2) → OPND(1)
15	ALPHA MAX.: THE ALPHA MAXIMUM OF OPND(1) AND OPND(2) → OPND(1)
16	TRANSFER: OPND(1) → RESULT

BUFFER FORMATS

General layout for Source Buffer Format (SBF), Working Buffer Format (WBF), and Target Buffer Format (TBF).

Column 1 - 1 word - SBF: temporary key field indicator for CF command

WBF: the value to be used for initialization after printing of a results field

TBF: print flag to associate printing of this field with a change of a BY field (BY processing table row number N)

Print flag = $2*N-1$ means print this field at top of BY number N

Print flag = $2*N$ means print this field at bottom of BY number N

Column 2 - 3 words - SBF: first word is data base format row number of key field for CF command.

WBF: four types of data: (1) alphanumeric characters representing field names, (2) \$Lbb in first word for integer literal in command line, (3) \$Tbb in first word for alphanumeric literal in command line, (4) \$Rbb in first word for calculation results

TBF: First word is row number of WBF of desired output field. Second word is key field indicator for CF command. Third word is data base format row number of key field for CF command.

Column 3 - 1 word - SBF, WBF, TBF: starting character for actual value in buffer being used. (In WBF, the data is actually in the command line instead of the working buffer if column 2 = \$Tbb)

Column 4 - 1 word - SBF, WBF, TBF: length of field (in characters)

Column 5 - 1 word - SBF, WBF, TBF: type of data in the field:

-1 means a binary integer contained in 4 characters

0 means an alphanumeric character string

1 means an integer in a numeric character string

2 means a date in YDDD numeric character string format

MULTILEVEL MOVE TABLE

Each row of this two-column table represents a move of data to be made by TFORMW subroutine.

Column 1 - 1 word - binary integer whose value is the row number of the Source Buffer Format array where information about the field in the Source Buffer is located.

Column 2 - 1 word - binary integer whose value is the row number of the Working Buffer Format array where information about the field in the Working Buffer is located.

MOVE TABLE CONTROL TABLE

Each row of this ~~three~~-column table defines which moves in the Multilevel Move Table are to be performed for the record in the Source Buffer from a particular data base level.

Column 1 - 1 word - binary integer whose value is the starting row number in the Multilevel Move Table.

Column 2 - 1 word - binary integer whose value is the number of rows in the Multilevel Move Table to be processed via TFORMW subroutine to get all the needed data transferred from the Source Buffer to the Working Buffer at a particular data base level.

Column 3 - 1 word - binary integer whose value is the format number for records at this data base level.

BY PROCESSING TABLE

Each successive row of this table defines a successively lower level subgroup of the input data and the processing associated with a change at that subgroup level.

Column 1 - 1 word - binary integer; index to Working Buffer Format (i.e., row number) pointing to the Grouping Field Name (GFN). If 0, it means the GFN was E&E. If <0 , then a calculation must be performed before a test for the BY change can be made.

Column 2 - 1 word - binary integer; starting row number of normal command table when column 1 is <0 .

Column 3 - 1 word - binary integer; number of rows of normal command table to be processed when column 1 is ≤ 0 .

Column 4 - 1 word - binary integer; starting row number of normal command table for use when the value of this BY field or calculation changes.

Column 5 - 1 word - binary integer; number of rows of normal command table to process when the value of this BY field or calculation changes.

Column 6 - 5 words - current value of the GFN for this subgroup level. An integer or calculation result is stored in the first word, whereas a text field may be all 20 characters.

